MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE D'ORAN
- MOHAMED BOUDIAF-

FACULTE DES MATHEMATIQUES ET D'INFORMATIQUE
DEPARTEMENT D'INFORMATIQUE

COURS

BASE DE DONNEES

DR. KHADIDJA BELBACHIR
MAITRE DE CONFERENCE B

Objectifs de l'enseignement : Ce cours devrait permettre à l'étudiant d'identifier l'intérêt de structurer et manipuler les données sous forme tabulaire. A travers le modèle relationnel et l'algèbre relationnelle sous-jacente orientés plus vers l'aspect pratique, l'étudiant devrait comprendre l'importance de structurer les données, le concept d'indépendance des données et des traitements, ainsi que l'intégrité et la cohérence des données.

Connaissances préalables recommandées : L'étudiant est sensé comprendre ce que c'est des fichiers (textes, binaires ou typés) et les avoir créés avec les langages préalablement étudiés.

Licence L2
Semestre 4

Table des matières

Liste des figures

INTRODUCTION GENERALE

Le présent polycopié s'adresse spécialement aux étudiants de deuxième année Licence en Informatique, et a pour objectif l'étude des principes et des techniques en matière de base de données. Ce cours permet aux étudiants d'avoir les prérequis et les notions de bases nécessaires sur les bases de données relationnelles et ce, après avoir disposé d'une vision globale sur les SGBD et les différents modèles de données.

Le contenu de polycopie est organisé en trois parties :

La première constitue une initiation aux bases de données et introduit les différents fonctionnalités et concepts de bases des SGBD. Cette partie décrit aussi les différents modèles de données et les règles de passage du modèle Entité/Association au modèle relationnel.

La deuxième partie traite les bases de données relationnelles, les notions fondamentales sont présentées : les dépendances fonctionnelles, la normalisation, décomposition d'une relation en 3^{éme} forme normale. De plus une interprétation du langage SQL pour la création et la manipulation des données est décrite.

Dans la troisième partie, l'accent est mis sur les opérateurs de l'Algèbre Relationnelle et leurs traductions en SQL pour l'interrogation et la manipulation des données.

Nous avons clôturé notre polycopié par une bibliographie qui englobe les ressources utilisées.

Chapitre 1 : Présentation des bases de données

1.1 Notion de fichiers (intérêts et limites)

L'utilisation de fichiers impose d'une part, à l'utilisateur de connaître l'organisation (séquentielle, indexée, ...) des fichiers qu'il utilise afin de pouvoir accéder aux informations dont il a besoin et, d'autre part, d'écrire des programmes pour pouvoir effectivement manipuler ces informations. En conséquence l'utilisation directe des fichiers soulève de très gros problèmes :

- Lourdeur d'accès aux données. En pratique, pour chaque accès, même le plus simple, il faudrait écrire un programme.
- Manque de sécurité. Si tout programmeur peut accéder directement aux fichiers, il est impossible de garantir la sécurité et l'intégrité des données.
- Pas de contrôle de concurrence. Dans un environnement où plusieurs utilisateurs accèdent aux mêmes fichiers, des problèmes de concurrence d'accès se posent.
- Duplication des données. Si plusieurs utilisateurs peuvent accéder aux fichiers, on peut avoir des données redondantes.

D'où le recours au concept de base de données pour prendre en charge ces problèmes.

1.2 Définition de base de données

Une base de données est un ensemble structuré de données apparentées qui modélisent un univers réel (administration, banque, université, hôpital, ...).
Plus précisément, une base de données informatisée est un ensemble structuré de données enregistrées sur des supports accessibles par l'ordinateur, représentant des informations du monde réel et pouvant être interrogées et mises à jour par une communauté d'utilisateurs.
Des exemples d'utilisation de base de données tels que :

- Gestion des personnels, étudiants, cours, inscriptions, ... de l'université.
- Système de réservation de places d'avion d'une compagnie aérienne.
- Gestion des comptes clients d'une banque.
- Gestion d'une bibliothèque.

1.2.1. Exemple d'une base donnée

On considère une compagnie aérienne. Celle-ci possède plusieurs services : comptabilité, réservation, affectation pilote, programmation des vols.

1.2.1.1 Approche classique (système de fichiers) :

Chaque service a son propre fichier, alors q' il y a des services qui manipulent les mêmes données.

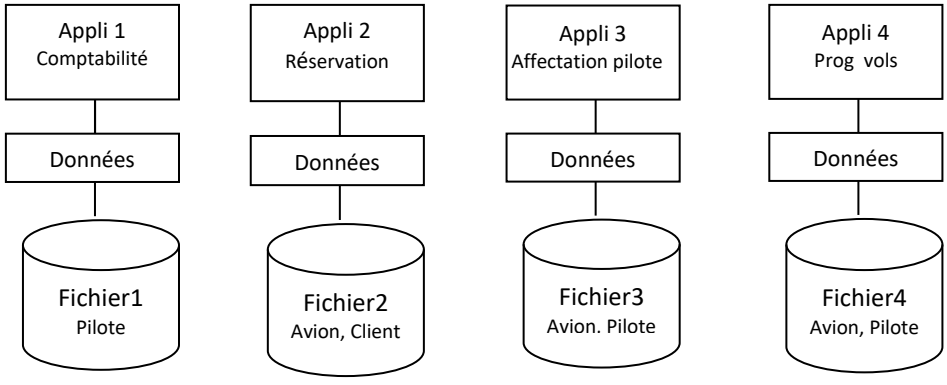


Figure 1. Approche classique

1.2.1.2 Approche BDD :

Tous les données sont partagées par tous les services (applications) .

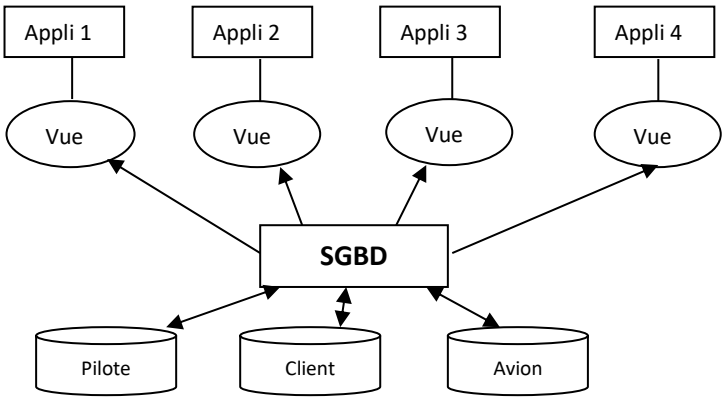


Figure 2. Approche BDD

1.2.2. Objectifs d'une base de données

Les principaux objectifs des bases de données :

- Eliminer la redondance et l'incohérence des données. Par exemple, le service comptabilité et affectation pilote utilisent les mêmes données Pilote.
- Independence, isoler l'utilisateur de toute modification au niveau de l'organisation physique, l'organisation logique et le schéma d'accès.
- Sécurité, sécuriser les données contre tout accès invalide ou modification illégale.
- Disponibilité, le temps de réponse ne doit pas être détérioré par l'utilisation en parallèle de la BD.

1.3 Définition de système de gestion de données

Un SGBD (système de gestion de base de données) est un système (ensemble de programmes) qui permet à l'utilisateur de créer et manipuler une BD partagée par plusieurs utilisateurs simultanément. Un SGBD doit permettre l'ajout, la modification et la recherche de données.

1.3.1 Fonctions et objectifs d'un SGBD

Le SGBD doit assurer les fonctions suivantes :

- Description des données, le SGBD offre à l'utilisateur un langage de description de données LDD.
- Manipulation des données, le SGBD fournit un langage permettant de rechercher, sélectionner et modifier les données. C'est un langage de manipulation de données LMD.
- Intégrité (Cohérence des données), Les données sont soumises à un certain nombre de contraintes d'intégrité (par exemple, le salaire doit être compris entre 20000 DA et 100000 DA). Ces contraintes doivent pouvoir être exprimées simplement et vérifiées automatiquement à chaque insertion, modification ou suppression des données.
- Confidentialité (Sécurité des données), les données doivent pouvoir être protégées contre les accès non autorisés. Pour cela, il faut pouvoir associer à chaque utilisateur des droits d'accès aux données.

- Reprise après panne, le SGBD offre des mécanismes de récupération des données comme la journalisation et les procédures de reprise après panne en cas d'incident matériel ou logiciel. Par exemple, sauvegarde de la base de données une fois par semaine.
- Partage des données, contrôle les accès concurrents entre plusieurs utilisateurs en même temps.
- Performances d'accès, permette d'obtenir des réponses aux requêtes en un temps raisonnable et de façon totalement transparente pour l'utilisateur.

1.3.2 Niveaux de représentation des données dans un SGBD

Trois niveaux de description des données ont été définis par la norme ANSI/SPARC.

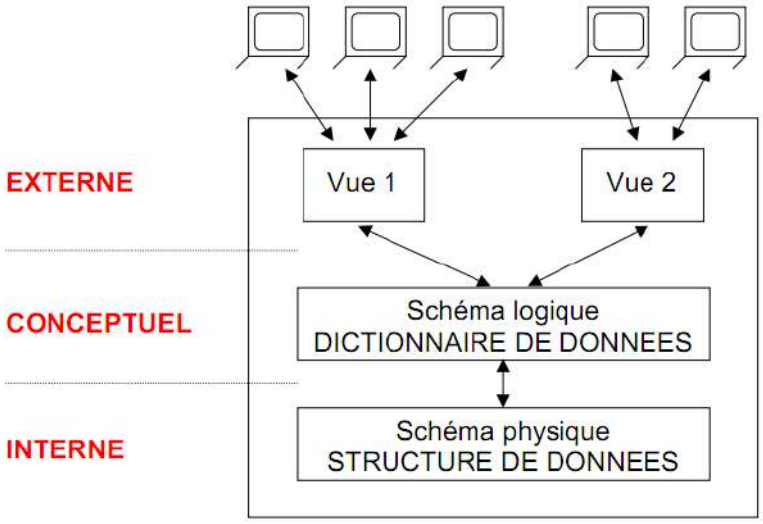


Figure 3. SGBD (Niveaux de représentation des données)

Le niveau externe : correspond à la perception de tout ou partie de la base par un groupe donné d'utilisateurs, on appelle cette description le *schéma externe* ou *vue*. Le niveau externe assure l'analyse et l'interprétation des requêtes et se charge également de convertir éventuellement les données brutes, issues de la réponse à la requête, dans un format souhaité par l'utilisateur (par exemple synthétiser les informations).

Le niveau conceptuel : contient la description des données selon le modèle opté (hiérarchique, réseau ou relationnel…). Dans le cas des SGBD relationnels, il s'agit d'une vision tabulaire exprimée en utilisant les concepts de relation, attributs et de contraintes d'intégrité. On appelle cette description le schéma conceptuel.

Le niveau interne : s'appuie sur un système de gestion de fichiers pour définir la politique de stockage ainsi que le placement des données. Le niveau physique est donc responsable du choix de l'organisation physique des fichiers ainsi que de l'utilisation de telle ou telle méthode d'accès en fonction de la requête. On appelle cette description le schéma interne.

1.3.3 Historique et quelques SGBD connus

L'apparition des premiers SGBD (conçus sur le modèle hiérarchique et réseau) a été les années 70. Ensuite, une deuxième génération et qui représente l'essentiel du marché BD aujourd'hui est apparue les années 80, ces SGBD sont basés sur le modèle relationnel. Aux années 90, la troisième génération

basée sur les systèmes à Objet a vu le jour. Ces SGBD sont fondés sur des modèles de données plus riches.

Il existe de nombreux systèmes de gestion de bases de données, tels que : PostgreSQL, MySQL, Oracle, Microsoft SQL, Sybase, Access ....

1.4. Modèles de données

Les différents modèles de BD sont souvent trop limités pour pouvoir représenter directement le monde réel. Une ou plusieurs modélisations intermédiaires sont donc utiles, le modèle E/A constitué l'une des premières et des plus courantes approches de conception.

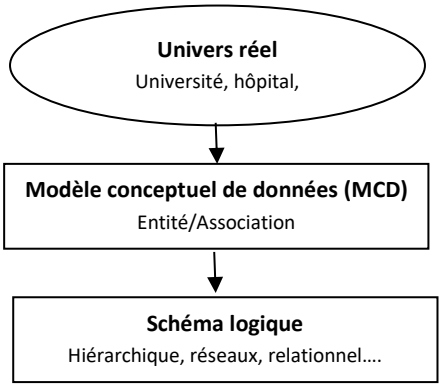


Figure 4. Modèles de données

1.4.1 Le modèle Entité/Association

Le modèle Entité/Association est un modèle conceptuel de données (MCD) de la méthode Merise, son schéma est représenté par des entités reliées par des associations. Le modèle E/A s'appuie sur trois concepts de base :

1. **Entité :** représente un objet matériel ou immatériel. Elle possède un nom, une liste d'attributs (propriétés) et un identifiant. Tel que, deux instances d'une entité ne puissent pas avoir le même identifiant.
2. **Association :** caractérisées par un nom, une liste des entités qui participent à l'association, une liste des attributs (éventuellement).
3. **Cardinalité :** précisent la participation de l'entité à l'association, d'une borne minimale et d'une borne maximale :
   - Minimale : nombre minimum de fois qu'une instance de l'entité participe aux instances de l'association, généralement 0 ou 1.
   - Maximale : nombre maximum de fois qu'une instance de l'entité participe aux instances de l'association, généralement 1 ou n.

EX :

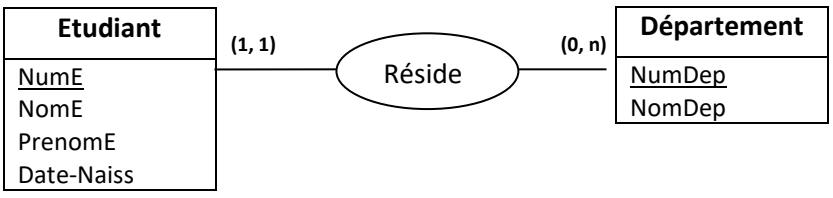


Figure 5. Exemple Entité/Association

- Un étudiant réside dans un (au minimum) et un seul département (au maximum), donc les cardinalités (Min, Max)= (1,1)
- Un département a pour résident aucun (au minimum) ou plusieurs étudiants (au maximum), donc les cardinalités (Min, Max) = (0, n)

1.4.2 Le Modèle hiérarchique

Ce type de modèle logique est représenté sous forme d'un arbre où les nœuds correspondent aux classes et les arcs aux associations.

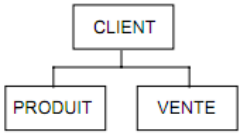


Figure 6. Modèle hiérarchique

1.4.3 Le Modèle réseau

Le schéma logique est représenté sous forme de graphe où les nœuds sont les classes et les arcs les associations.

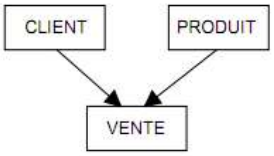


Figure 7. Modèle réseau

1.4.4 Le Modèle relationnel

Le schéma relationnel est l'ensemble des relations qui modélisent le monde réel. Les relations représentent les entités du monde réel (comme des clients, des produits, ….) ou les associations entre ces entités (vente).
Chaque relation a une clé primaire permettant d'identifier les données, et éventuellement des clés étrangères faisant des références vers les données des autres relations.

Exemple :
Client (IdCli, nom, ville)
Produit (IdPro, nom, prix, qstock)
Vente (#IdCli, #IdPro, date, qte)

Où IdCli et IdPro sont les clés primaires de Client et Produit respectivement. La relation Vente a deux clés étrangères (IdCli et IdPro) et une clé primaire composée (IdCli, IdPro).

1.4.5 Les règles de passage du modèle E/A au relationnel

Il existe un certain nombre de règles :

Règle 1 : Chaque entité est transformée en relation et son identifiant devient clé primaire.

Règle 2 : Association de type (?,1)-(?,n), l'identifiant de l'entité forte (côté N) est ajouté comme clé étrangère à la relation de l'entité faible (côté 1).

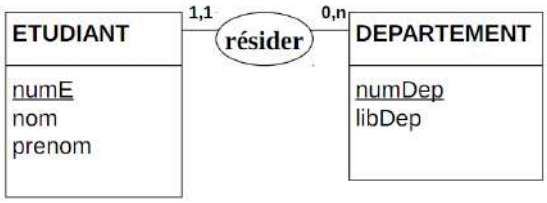


Figure 8. Exemple de la règle 2 de passage du modèle E/A au relationnel

ETUDIANT (numE, nom, prénom, #numDep)
DEPARTEMENT (numDep, libDep)

Règle 3: Association de type (?,n)-(?,n), l'association devient une nouvelle relation dont la clé primaire (clé composée) est l'ensemble des identifiants des entités participant à l'association. Tout attribut de l'association devient attribut de la nouvelle relation.

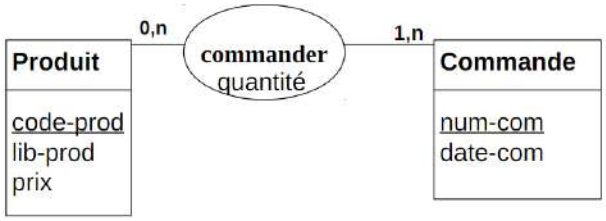


Figure 9. Exemple de la règle 3 de passage du modèle E/A au relationnel

PRODUIT (code-prod, lib-prod, prix)
COMMANDE (num-com, date-com)
LIGNE-DE-COMMANDE (#code-prod, #num-com, quantité)

Règle 4 : Association de type (?,1)-(?,1), l'identifiant de l'une de entités faibles est ajouté comme clé étrangère à la relation de l'entité associée et tout attribut de l'association devient attribut de cette relation.

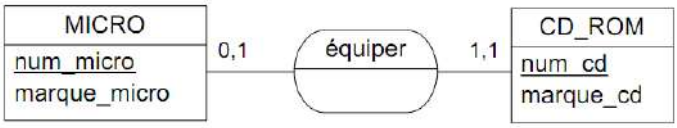


Figure 10. Exemple de la règle 4 de passage du modèle E/A au relationnel

MICRO (Num_Micro, Marque_Micro)
CD_ROM (Num_Cd, Marque_Cd, #Num_Micro)

Chapitre 2 : Modèle relationnel

2.1 Définition du modèle relationnel

Un des grands avantages du modèle relationnel est sa très grande simplicité. Il n'existe en effet qu'une seule structure, la relation. Une relation peut simplement être représentée sous forme de table.

2.2 Concepts de base

2.2.1 Le domaine

On appel domaine tout ensemble de valeurs atomiques d'un certain type sémantique.
Ex : COULEUR = {blanc, rouge, vert}
Deux ensembles peuvent avoir les mêmes valeurs bien que sémantiquement distincts
Ex : NUM_ELV = { 1, 2, … , 2000 }
NUM_ANNEE = { 1, 2, … , 2000 }

2.2.2 La relation

La relation R est un sous ensemble du produit cartésien de plusieurs domaines :

$$R \subset D_1 \times D_2 \times ... \times D_n$$

Où D_1, D_2, ... , D_n sont les domaines de R
EX:

Les domaines :
NOM= { Mokran, Osmani }
PRENOM= { farid, Omar, Fatima }
DATE_NAISS = {Date entre 1/1/1990 et 31/12/2020}
La relation ELEVE :
ELEVE ⊂ NOM × PRENON × DATE_NAISS
ELEVE = { (Mokran, Farid, 1/1/1992), (Osmani, Fatima, 2/2/1994) }

2.2.3 Les n_uplets

Un élément d'une relation est un n-uplet de valeurs (tuple en anglais), un n-uplet représente un fait.
EX: Mokran Farid est un élève né le 1 janvier1992.

n-uplet →

NOM	PRENON	DATE_NAISS
Mokran	Farid	1 janvier1992
⋮	⋮	⋮

Figure 11. n_uplets

2.2.4 Les attributs

Chaque composante (colonne) d'une relation est un attribut, le nom donné à un attribut est porteur de sens.
EX : NOM

2.2.5 Schéma de relation

Un schéma de relation est simplement son nom suivi de la liste des attributs :

$$R(A_1, A_2, ..., A_n)$$

Ex:
ELEVE (NOM, PRENOM, NAISS)

2.2.6 Schéma d'une base de données relationnelle

Le schéma d'une base de données est défini par l'ensemble des schémas des relations qui la composent.

2.3 Normalisation

Consiste à créer des schémas relationnels répondant à un certain standard appelé forme normale qui devront respecter les contraintes de dépendances. Les règles de normalisation permettent d'assurer le non-redondance des données, l'intégrité des données et la facilité de mise à jour.

Exemple : pour représenter des personnes, ainsi que les véhicule qu'elles ont acheté. On peut créer la relation Personne(NSS, Nom, Prénom, Marque, Type, Puiss, Date, Prix) remplie par les données suivantes :

NSS	Nom	Prénom	Marque	Type	Puiss	Date	Prix
16607	Mokran	Farid	Renaut	Clio	5	01/01/2008	14000
16607	Mokran	Farid	Peugeot	207	7	02/07/2006	11000
24856	Osmani	Fatima	Peugeot	207	7	01/10/2009	14000
15753	Amrani	Omar	Peugeot	207	7	08/08/2012	15000
15753	Amrani	Omar	Renaut	Clio	5	07/06/2011	16000
15753	Amrani	Omar	BMW	X6	10	04/05/2014	905000

Figure 12. Exemple de redondance

On peut alors constater que des redondances sont présentes. Par conséquence, ces redondances produisent des problèmes de contrôle de la cohérence de l'information (erreur dans la saisie d'un numéro de sécurité sociale), de mise à jour (changement de nom à reporter dans de multiples tuples), de perte d'information lors de la suppression de données (disparition des informations concernant un type de véhicule).

Le processus de normalisation permet de décomposer une relation non normalisée en un ensemble équivalent de relations normalisées (en 3ème forme normale).

2.3.1 Dépendance fonctionnelle

Soit R(A1, A2, ..., An) un schéma de relation, et Soit X et Y des sous-ensembles de {A1,A2,...An}.
On dit que Y dépend fonctionnellement de X (ou X détermine Y), notée (X->Y) si : $\forall$ t1 et t2, deux tuples de R, si t1[x]=t2[x] alors t1[y]=t2[y], pour la même valeur de X correspond toujours une même valeur de Y.
Ex: PRODUIT (no_prod, nom, prixUHT)
no_prod → (nom, prixUHT)

3.3.1.1 Propriétés des DF

Soit X, Y, Z et W des ensembles d'attributs, les DF se soumettent à des propriétés mathématiques particulières, dites axiomes d'Armstrong :

- **Réflexivité :** Si Y ⊆ X alors X→Y et donc X→X
- **Augmentation :** si X→Y alors XZ→YZ
- **Transitivité :** si X→Y et Y→Z alors X→Z

A partir des axiomes d'Armstrong, on peut déduire un certain nombre de propriétés supplémentaires :

- **Pseudo-transitivité :** si X→Y et WY→Z alors WX→Z (par augmentation et réflexivité)
- **Union :** si X→Y et X→Z alors X→YZ (par réflexivité, l'augmentation et la transitivité)
- **Décomposition :** si X→YZ alors X→Z et X→Y (par réflexivité et transitivité)

3.3.1.2 Dépendance fonctionnelle élémentaire (DFE)

Une DF X -> A est élémentaire si A est un attribut unique non inclus dans X et il n'existe pas de X' inclus dans X tel que X' →A (∄ X' ⊆ X : X'→A).

Ex :

- AB→C est élémentaire si (∄ A→C et B→C).
- Nom, DateNaissance, LieuNaissance→Prénom est élémentaire.
- AB→A n'est pas élémentaire car A est incluse dans AB.
- AB→CB n'est pas élémentaire car CB n'est pas un attribut, mais un groupe d'attributs.
- N°SS→Nom, Prénom n'est pas élémentaire car (Nom, Prénom) n'est pas un attribut unique.

2.3.2 La clé d'une relation

C'est un attribut (ou groupe minimum d'attributs) qui détermine tous les autres.

Ex : PRODUIT (no_prod, nom, prixUHT)

no_prod → (nom, prixUHT)

no_prod est une clé

Une relation peut posséder plusieurs clés, on les appelle clés candidates. Par exemple dans la relation PRODUIT, nom est une clé candidate.

Il existe deux types de clé :

- **Clé primaire :** choix d'une clé parmi les clés candidates
- **Clé étrangère ou clé secondaire :** un attribut (ou groupe d'attributs) qui fait référence à la clé primaire d'une autre relation.

  Ex:

  ETUDIANT (numE, nom, prénom, #numDep)

  DEPARTEMENT (numDep, libDep)

  numDep dans ETUDIANT est une clé étrangère.

2.3.3 Les règles d'intégrité

Les règles d'intégrité sont les assertions qui doivent être vérifiées par les données contenues dans une BD. Le modèle relationnel impose les contraintes suivantes :

- **Intégrité de domaine :** contrôle des valeurs des attributs et entre valeurs des attributs.
- **Intégrité de clé :** Les valeurs de clés primaires doivent être uniques et non NULL.
- **Intégrité révérencielle :** Les valeurs de clés étrangères sont 'NULL' ou sont des valeurs de la clé primaire auxquelles elles font référence. Ces contraintes permettent d'éviter les anomalies de mises à jour.

2.3.4 Les formes normales

Les formes normales ont pour objectif de définir la décomposition des schémas relationnels en se basant sur les dépendances fonctionnelles entre les attributs d'une relation.

Pour assurer la cohérence et le non-redondance des données, toutes les relations de la base de données doivent être en 3ème Forme normale.

2.3.4.1 1ère Forme Normale (1FN)

Une relation est en 1FN si tout attribut est atomique (non décomposable)

Contre-exemple

ELEVE (no_elv, nom, prénom, liste_notes)

Un attribut (liste_notes) ne peut pas être un ensemble de valeurs. Par exemple : (1, Mokran, Farid, (15, 12, 14))

Solution : la décomposition

ELEVE (no_elv, nom, prénom)

NOTE (no_elv, no matière, note)

2.3.4.2 2ème Forme Normale (2FN)

Une relation est en 2FN si elle est en 1FN, et si tout attribut n'appartenant pas à la clé ne dépend pas d'une partie de la clé.

Contre-exemple

Une relation en 1FN qui n'est pas en 2FN car la clé = (date, no_cli, no_pro), et le prixUHT ne dépend que de no_pro :

COMMANDE (date, no_cli, no_pro, qte, prixUHT)

DF= {date, no_cli, no_pro → qte, date, no_cli, no_pro → prixUHT, no_pro→ prixUHT}

Solution : la décomposition

COMMANDE (date, no_cli, no_pro, qte)

PRODUIT (no_pro, prixUHT)

2.3.4.3 3ème Forme Normale (3FN)

Une relation est en 3FN si elle est en 2FN, et si tout attribut n'appartenant pas à la clé ne dépend pas d'un attribut non clé.

Ceci correspond à la non-transitivité des D.F ce qui évite les redondances.

Contre-exemple

Une relation en 2FN qui n'est pas en 3FN car la clé=matricule, et la puissance dépend de (marque, modèle) :

VOITURE (matricule, marque, modèle, puissance)

DF= {matricule→ marque, matricule→ modèle, matricule→ puissance, (marque, modèle) → puissance}

Solution : la décomposition

VOITURE (matricule, marque, modèle)

MODELE (marque, modèle, puissance)

2.3.4.4 3ème Forme Normale de BOYCE-CODD (BCNF)

Une relation est en BCNF si elle est en 3NF et si tout attribut qui n'appartient pas à une clé n'est pas source d'une DF vers une partie d'une clé. C'est à dire que les seules DFE existantes sont celles dans lesquelles une clé détermine un attribut.

La forme normale de Boyce-Codd permet d'éliminer les dépendances entre les attributs n'appartenant pas à une clé vers les parties de clé.

Contre-exemple

Une relation en 3FN qui n'est pas BCNF :

CODEPOSTAL (ville, rue, code)

DF= {(ville, rue) → code, code → ville}

On vérifie qu'elle est en 3FN, elle n'est pas BCNF car la clé = (ville, rue) (ou (code, ville) ou (code, rue)), et il existe une DFE qui n'est pas issue d'une clé et qui détermine un attribut appartenant à une clé (code → ville).

2.3.5 Fermeture transitive

On appelle fermeture transitive F^+ d'un ensemble F, l'ensemble de DFE enrichi de toutes les DFE déduites par transitivité.

F+= {DFE} U {DFEs transitives}

EX :

Soit l'ensemble F = {A→B, B→C, B→D, A→E}.

La fermeture transitive de F est F^+ = F U {A→C, A→D}

Donc F^+ = { A→B, B→C, B→D, A→E, A→C, A→D }

2.3.5.1 Algorithme du calcul de la fermeture d'un ensemble d'attributs

La fermeture d'un ensemble d'attributs X, $(X)^+$ représente l'ensemble des attributs de la relation R qui peuvent être déduits de X à partir d'un ensemble F de DF et des règles d'inférences.

Y est inclus dans $(X)^+$ si et seulement si X --> Y.

Algorithme :

1. Initialisation de $(X)^+$ à X
2. trouver une DF dans F dont la partie gauche est incluse dans $(X)^+$
3. ajouter dans $(X)^+$ la partie droite de cette DF
4. répéter les opérations 2 et 3 jusqu'à ce que $(X)^+$ n'évolue plus

2.3.5.2 Exemple de déroulement

Soit l'ensemble de DF suivant : F = {A → D, AB → E, BI → E, CD → I, E → C}

Calculer A^+, AB^+ et BI^+ sous F, en appliquant l'algorithme de la fermeture d'un ensemble d'attributs.

Calculer A^+ :

1. A^+= {A}
2. A → D
3. A^+= {A, D}
4. A^+= {A, D}

A^+ n'évolue pas, l'algorithme s'arrête et A^+= {A, D}

Calculer AB^+ :

1. AB^+= {A, B}
2. AB → E
3. AB^+= {A, B, E}
2. A → D
3. AB^+= {A, B, E, D}
2. E → C
3. AB^+= {A, B, E, D, C}

4. AB^+= {A, B, E, D, C}

AB^+ n'évolue pas, l'algorithme s'arrête et AB^+= {A, B, E, D, C}

Calculer BI^+:

1.BI^+= {B, I}

2. BI → E

3. BI^+= {B, I, E}

2. E → C

3. BI^+= {B, I, E, C}

4. BE^+= {B, I, E, C}

BI^+ n'évolue pas, l'algorithme s'arrête et BI^+= {B, I, E, C}

2.3.6 Couverture minimale

La couverture minimale (graphe minimum) de F (ensemble de DF) est un ensemble minimal de DFE permettant de générer toutes les autres.

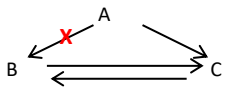
- Aucune dépendance dans F n'est redondante (pour toute DF f de F, F-{f} n'est pas équivalent à F), sont des DFE non déduites. Par exemple, E→D est déduite de E→C et C→D.
- Toute DFE des attributs est dans la fermeture transitive de F, c'est un sous-ensemble minimal de la Fermeture Transitive.
  Tout ensemble de DFE (et donc tout ensemble de DF) admet au moins une couverture minimale (et en pratique souvent plusieurs).

EX :

L'ensemble F={A→B, A→C, B→C, C→B} admet les deux couvertures minimales :

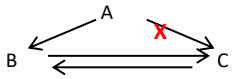
CM1 = {A→C, B→C, C→B}

GM1 :



CM2 = {A→B, B→C, C→B}

GM2 :



2.3.6.1 Algorithme de la couverture minimale

Entrée : F (un ensemble de dépendances fonctionnelles)
Sortie : CM (la couverture minimale)
1. CM ← F
2. Remplacer chaque DF ($X \rightarrow [A_1;...; A_n]$) dans CM par n DF ($X \dashrightarrow A_i$)
3. Pour chaque DF ($X \rightarrow A$) dans CM Faire
   Si $\exists B \subset X / A \in B^+$ alors
   $CM = CM - \{X \rightarrow A\} \cup \{B \rightarrow A\}$
   FinSi
   FinPour

4. Pour chaque DF ($X \rightarrow A$) ∈ CM faire

Calculer $(X)^+$ dans CM – {X→A} ;
Si A ∈ $(X)^+$ alors
X→A est redondante
CM = CM − {X→A}
FinSi
FinPour

2.3.6.2 Exemple de déroulement

Soit la relation R (A, B, C, D, E, F) satisfaisant l'ensemble F de dépendances fonctionnelles suivant :
F = {AB → C, C → A, BC → D, D → EF, CE → AF, ABE → C}
Calculer CM la couverture minimale de F.

1. CM= {AB → C, C → A, BC → D, D → EF, CE → AF, ABE → C}

2***. Mettre les dépendances de CM sous forme canonique :***

CM= {AB → C, C → A, BC → D, D → E, D → F, CE → A, CE → F, ABE → C}

3. Supprimer les dépendances non élémentaires :

Les dépendances (C → A, D → E, D → F) sont élémentaires, car leurs déterminants est un seul attribut.

Il reste à vérifier les dépendances (AB → C, BC → D, CE → A, CE → F, ABE → C} en utilisant l'algorithme de calcul de la fermeture d'un ensemble d'attributs :

A^+={A} B^+={B} C^+= {C, A} E^+={E} AB^+= {A, B, C, D, F, E} AE^+={A, E} BE^+={B, E}

- AB → C ?

  C∉A^+ et C∉B^+, donc (AB → C) est élémentaire.

- BC → D ?

  D∉B^+ et D∉C^+, donc (BC → D) est élémentaire.

- CE → A ?

  A∈C^+, donc (CE → A) n'est pas élémentaire et sera remplacée par C → A :

  (CM = CM − {CE → A} ∪ {C → A})

- CE → F ?

  F∉C^+ et F∉E^+, donc (CE → F) est élémentaire.

- ABE → C ?

  C∈AB^+, donc (ABE → C) n'est pas élémentaire et sera remplacée par AB → C :

  (CM = CM − {ABE → C} ∪ {AB → C})

Donc, CM= {AB → C, C → A, BC → D, D → E, D → F, CE → F}

4. Supprimer les dépendances redondantes :

- AB → C ?

  Calculer AB^+ dans CM-{AB → C} : AB^+={A, B}

  C∉ AB^+, donc (AB → C) n'est pas redondante.

- C → A ?
  Calculer C^+ dans CM-{C → A} : C^+={C}
  A∉ C^+, donc (C → A) n'est pas redondante.

- BC → D ?
  Calculer BC^+ dans CM-{BC → D} : BC^+={B, C, A}
  D∉ BC^+, donc (BC → D) n'est pas redondante.

- D → E ?
  Calculer D^+ dans CM-{D → E} : D^+={D, F}
  E∉ D^+, donc (D → E) n'est pas redondante.

- D → F ?
  Calculer D^+ dans CM-{D → F} : D^+={D, E}
  F∉ D^+, donc (D → F) n'est pas redondante.

- CE → F ?
  Calculer CE^+ dans CM-{CE → F} : CE^+={C, E, A}
  F∉ CE^+, donc (CE → F) n'est pas redondante.

Finalement, CM= {AB → C, C → A, BC → D, D → E, D → F, CE → F}

2.3.7 Décomposition d'une relation

L'algorithme de décomposition identifie les bonnes clés du schéma relationnel à partir de **la couverture minimale**, cette dernière est extraite à partir de **la fermeture transitive** en utilisant les dépendances fonctionnelles et les Axiomes de transitivité.

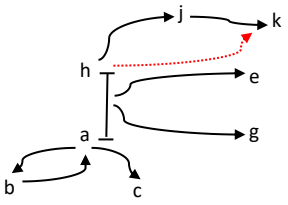
a. Algorithme de décomposition en 3ième forme normale

L'algorithme prend en entrée une relation universel R soumis à F (les dépendances fonctionnelles) et donne en résultat un ensemble $\{R_i(\underline{U_i},F_i)\}$ tel que tout R_i est en 3FN.

1. Calculer une couverture minimale (CM) à partir de l'ensemble de toutes les dépendances fonctionnelles élémentaires de F
2. Trouver les clés candidates
3. Regrouper les DF ayant même partie gauche G_j -> D_j.
4. Générer les relations maximaux R_i={U_i=G_j ∪ D_j ; G_j -> D_j}
5. Si aucune clé ne figure dans une des relations produites à l'étape précédente ajouter une relation R_i = {K; K->K}

b. Exemple

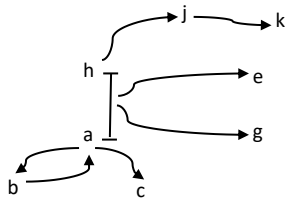
Soit la relation R = {a, b, c, d, e, f, g, h, j, k} avec l'ensemble de DF F= {a→b; a→c; a,h→e; a,h→g; h→j; j→k; h→k; b→a}



Graphe de F

1. Couverture minimale (suppression des DF redondantes) : h→k est redondante car elle peut être obtenue avec h→j et j→k par transitivité.

   CM={a→b ; a→c ; a,h→e; a,h→g; h→j ; j→k ; b→a}



Graphe minimum

2. Clés candidats : (a,h)

3. Regroupement des DF :
   E_1 = { a→b ; a→c} ; E_2 = { a,h→e ; a,h→g } E_3 = { h→j} ; E_4 = { j→k} ; E_5 = { b→a}
4. Génération des relations :
   R_1= {a,b,c}
   R_2= {a,h,e,g}
   R_3= {h,j}
   R_4= {j,k}
   R_5= {b, a}

c. Décomposition sans perte

La décomposition sans perte d'information consiste à ajouter une relation contenant la clé, même si la clé figure dans l'une des relations produites.
Dans l'exemple si on a choisi **(a,h)** comme clé, il faut ajouter la relation R_6={a,h}

2.4 Modèle relationnel logique (SQL)

Le modèle logique est un modèle formel, qui spécifie les données telles qu'elles vont exister dans l'application informatique.
Dans le modèle relationnel, les données sont représentées par des tables, des colonnes et des lignes :

- Tables (suite de lignes) : relations
- Colonnes (Chaque valeur est typée) : attributs
- Lignes (suite de valeurs) : éléments ou n-uplets (tuples)

Ex : Table Voiture

Colonne ↓

Code	Type	Marque	Puissance
1	Espace	Renaut	100
2	Clio	Renaut	60
3	Megane	Renaut	80
4	206	Peugeot	60
5	106	Peugeot	45

Ligne →

Figure 13. Table Voiture

2.4.1 Description de SQL (Structured Query Language)

Le langage SQL (*langage de requête structuré*) peut être considéré comme un langage de définition de données (LDD) et langage de manipulation de données (LMD). Le langage SQL permet de façon générale la définition, la manipulation et le contrôle de sécurité de données.

2.4.2 Définition de données

Le LDD permet de créer, modifier, supprimer des données. Il permet également de définir le domaine des données (nombre, chaîne de caractères, date, booléen…) et d'ajouter des contraintes de valeur sur les données. Il permet enfin d'autoriser ou d'interdire l'accès aux données pour un utilisateur donné.

i. Création de table

La commande CREATE TABLE permet de créer une table en SQL. La création d'une table sert à définir les colonnes et le type de données qui seront contenus dans chacun des colonnes (entier, chaîne de caractères, date, valeur binaire …).

Syntaxe :
```
CREATE TABLE nom_de_la_table
(colonne1 type_donnees,
colonne2 type_donnees,
colonne3 type_donnees,
……..)
```

Il est également possible de définir des options telles que :

- NOT NULL : empêche d'enregistrer une valeur nulle pour une colonne.
- DEFAULT : attribuer une valeur par défaut si aucune donnée n'est indiquée pour cette colonne lors de l'ajout d'une ligne dans la table.
- PRIMARY KEY : indiquer si cette colonne est considérée comme clé primaire.
- FOREIGN KEY : indiquer si cette colonne est considérée comme clé étrangère.

Exemple :

- Création de la table voiture :
```
CREATE TABLE Voiture
(Code INT PRIMARY KEY NOT NULL,
Type VARCHAR(20),
Marque VARCHAR(20),
Puissance INT)
```

- Création de la table personne :
```
CREATE TABLE Personne
(Nss INT PRIMARY KEY NOT NULL,
Nom VARCHAR(30),
Prenom VARCHAR(30),
Voiture INT DEFAULT 0,
FOREIGN KEY(Voiture) REFERENCES Voiture(Code))
```

ii. Modification de schéma

a. ALTER

La commande ALTER TABLE permet de modifier une table existante. Il est ainsi possible d'ajouter une colonne, d'en supprimer une ou de modifier une colonne existante, par exemple pour changer le type :

ALTER TABLE nom_table instruction

- **Ajouter une colonne :**

  ALTER TABLE nom_table ADD nom_colonne type_donnees

  EX : ALTER TABLE utilisateur ADD adresse_rue VARCHAR(255)

- **Supprimer une colonne :**
  ALTER TABLE nom_table DROP nom_colonne
- **Modifier une colonne :**
  ALTER TABLE nom_table MODIFY nom_colonne type_donnees
- **Renommer une colonne :**
  ALTER TABLE nom_table CHANGE colonne_ancien_nom colonne_nouveau_nom type_donnees

b. DROP

La commande DROP TABLE permet de supprimer définitivement une table d'une base de données. Cela supprime en même temps les éventuels index, contraintes et permissions associées à cette table.
Syntaxe : DROP TABLE nom_table
Ex : DROP TABLE Voiture

2.4.3 Manipulation des données

Le LMD est l'ensemble des commandes concernant la manipulation des données dans une base de données. Le LMD permet l'ajout, la suppression et la modification de lignes, la visualisation du contenu des tables et leur verrouillage.

i. INSERT

L'insertion de données dans une table s'effectue à l'aide de la commande INSERT INTO. Cette commande permet au choix d'inclure une seule ligne à la base existante ou plusieurs lignes d'un coup.

- **Insertion d'une ligne à la fois :**
  INSERT INTO nom_table VALUES ('valeur 1', 'valeur 2', ...)
- **Insertion de plusieurs lignes à la fois :**
  INSERT INTO Voiture (Code, Type, Marque, Puissance) VALUES
  (1, 'Espace', 'Renaut', 100),
  (2, 'Clio', 'Renaut', 60),
  (3, 'Megane', 'Renaut', 80) ;

ii. UPDATE

La commande UPDATE permet d'effectuer des modifications sur des lignes existantes. Très souvent cette commande est utilisée avec WHERE pour spécifier sur quelles lignes doivent porter la ou les modifications.

Syntaxe : UPDATE nom_table
SET colonne_1 = 'valeur 1', colonne_2 = 'valeur 2',.....
WHERE condition ;

Cette syntaxe permet d'attribuer une nouvelle valeur à la colonne nom_colonne pour les lignes qui respectent la condition stipulée avec WHERE.

Ex : UPDATE Voiture
SET Puissance = 80
WHERE Type='Clio' ;

iii. DELETE

La commande DELETE permet de supprimer des lignes dans une table. En utilisant cette commande associée à WHERE il est possible de sélectionner les lignes concernées qui seront supprimées.

Syntaxe : DELETE FROM nom_table
WHERE condition

Ex : DELETE FROM Voiture
WHERE Type='Megane'

Chapitre 3 : Algèbre Relationnelle

3.1 Définition

- L'algèbre relationnelle consiste en un ensemble d'opérations qui permettent de manipuler des relations, on peut ainsi faire l'union ou la différence de deux relations, sélectionner une partie de la relation, effectuer des produits cartésiens ou des projections, etc.
- Ensemble d'opérateurs qui s'appliquent aux relations et produisant une relation en résultat.
- Une requête est une expression algébrique qui s'applique à un ensemble de relations et produit une relation finale (le résultat de la requête). On peut voir l'algèbre relationnelle comme un langage de programmation très simple qui permet d'exprimer des requêtes sur une base de données relationnelle.
- Les opérateurs peuvent être classifiés en opérateurs unaires et opérateurs binaires :
  - Unaires : sélection (restriction), projection,
  - Binaires : union, intersection, différence, produit cartésien, jointure, division

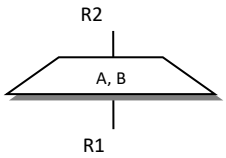
3.2 Les opérateurs unaires

Les opérateurs unaires qui sont les plus simples (sélection, projection) et qui portent sur une seule relation.

3.2.1 Projection

La projection d'une relation R1 est la relation R2 obtenue en supprimant les attributs de R1 non mentionnés. Elle correspond à un découpage vertical.
On notera : R2 = $\pi_{(Ai, Aj, \dots, Am)}$R1
la projection d'une relation R1 sur les attributs Ai, Aj, … , Am
EX :

R2= $\pi_{(A, B)}$ R1



R1	A	B	C
	a_1	b_1	c_1
	a_2	b_1	c_2
	a_1	b_1	c_1
	a_3	b_1	c_3
	a_4	b_2	c_3
	a_5	b_1	c_4

R2	A	B
	a_1	b_1
	a_2	b_1
	a_1	b_1
	a_3	b_1
	a_4	b_2
	a_5	b_1

Figure 14. Projection

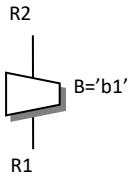
3.2.2 Sélection

La sélection d'une relation R1 est une relation R2 de même schéma n'ayant que les n-uplets de R1 répondant à la condition énoncée. Elle correspond à un découpage horizontal.
On notera : R2 = $\sigma_{(condition)}$R1

Où **condition** est une relation d'égalité(=) ou d'inégalité (≤, ≥, ≠, <, >) entre 2 attributs ou entre un attribut et une valeur.

EX :

R2= $\sigma_{(B='b1)'}$ R1



R1	A	B	C
	a_1	b_1	c_1
	a_2	b_1	c_2
	a_1	b_2	c_1
	a_3	b_1	c_3
	a_4	b_2	c_3
	a_5	b_1	c_4

R2	A	B	C
	a_1	b_1	c_1
	a_2	b_1	c_2
	a_3	b_1	c_3
	a_5	b_1	c_4

Figure 15. Sélection

3.2.3 Traduction en SQL

a. Projection par requêtes simples (SELECT-FROM)

L'utilisation la plus courante de SQL consiste à lire des données de la BD. Cela s'effectue grâce à la commande SELECT, qui retourne des lignes dans une table résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table par projection.

Syntaxe : SELECT [ALL/DISTINCT] nom_colonne1, nom_colonne2, …..
FROM nom_table

- L'option ALL, l'option par défaut. Elle permet de sélectionner l'ensemble des lignes satisfaisant à la condition.
- L'option DISTINCT permet de ne conserver que des lignes distinctes, en éliminant les doublons.

EX : R2= $\pi_{(A, B)}$ R1

SELECT A, B
FROM R1

R2	A	B
	a_1	b_1
	a_2	b_1
	a_1	b_1
	a_3	b_1
	a_4	b_2
	a_5	b_1

SELECT DISTINCT A, B
FROM R1

R2	A	B
	a_1	b_1
	a_2	b_1
	a_3	b_1
	a_4	b_2
	a_5	b_1

Figure 16. Projection par requêtes simples

b. Sélection (clause WHERE)

En SQL, les sélections s'expriment à l'aide de la clause *WHERE* suivie d'une condition logique exprimée à l'aide :

- Des opérateurs arithmétiques : =, >, <, !=, <=, >=, +, -, /, * …..

- Des opérateurs logiques : AND, OR, NOT
- Des comparateurs de chaines : IN, BETWEEN, LIKE, IS NULL ….

Syntaxe : SELECT nom_colonne
FROM nom_table
WHERE condition

EX : $R2 = \sigma_{(B='b1)'} R1$

```
SELECT *
FROM R1
WHERE B='b1'
```

c. *Tri de résultats (ORDER BY)*

La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.

Syntaxe :
```
SELECT colonne1, colonne2, colonne3…
FROM table
ORDER BY colonne1 DESC, colonne2 ASC …
```
Par défaut les résultats sont classés par ordre ascendant ASC, toutefois il est possible d'inverser l'ordre en utilisant le suffixe DESC après le nom de la colonne.

EX:

Voiture	Code	Type	Marque	Puissance
	1	Espace	Renaut	100
	2	306	Peugeot	80
	3	Clio	Renaut	60
	4	Megane	Renaut	80
	5	206	Peugeot	60
	6	106	Peugeot	45

Figure 17. Exemple avant Tri de résultats

En utilisant deux méthodes de tri, il est possible de retourner les Marques par ordre alphabétique et pour chaque marque, trier la Puissance par ordre décroissant :

```
SELECT *
FROM Voiture
ORDER BY Marque ASD, Puissance DESC
```

Voiture	Code	Type	Marque	Puissance
	2	306	Peugeot	80
	5	206	Peugeot	60
	6	106	Peugeot	45
	1	Espace	Renaut	100
	4	Megane	Renaut	80
	3	Clio	Renaut	60

Figure 18. Exemple après Tri de résultats

3.3 Les opérateurs binaires ou n-aires

Ce sont les opérateurs binaires ensemblistes (union, intersection, différence) entre deux relations, ou les opérateurs binaires ou n-aires (produit cartésien, jointure, division) entre deux ou plusieurs relations (tables).

3.3.1 Union

L'union de deux relations R1 et R2 de même schéma est une relation R3 de schéma identique contenant tous les tuples existant dans l'une ou l'autre des relations R1 et R2.
On notera : R3 = R1 ∪ R2

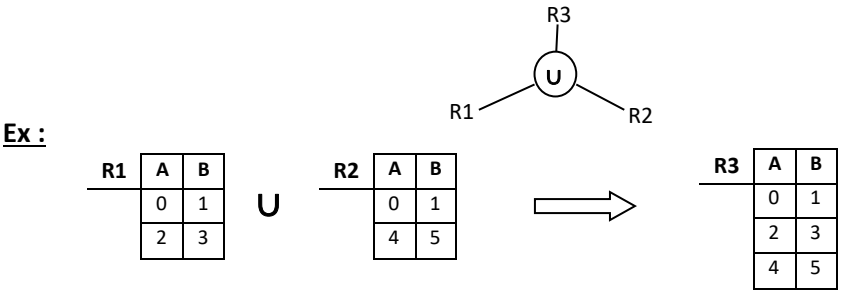


Figure 19. Exemple opérateur Union

3.3.2 Intersection

L'intersection entre deux relations R1 et R2 de même schéma est une relation R3 de schéma identique contenant les tuples appartenant à R1 et R2.
On notera : R3 = R1 ∩ R2

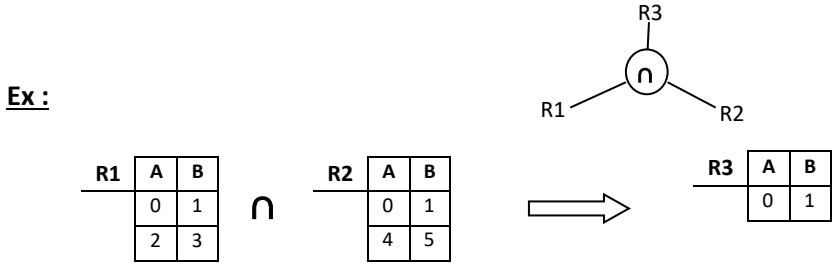


Figure 20. exemple opérateur Intersection

3.3.3 Différence

La différence entre deux relations R1 et R2 de même schéma est une relation R3 de schéma identique contenant les tuples de R1 n'appartenant pas à R2.
On notera : R3 = R1 − R2

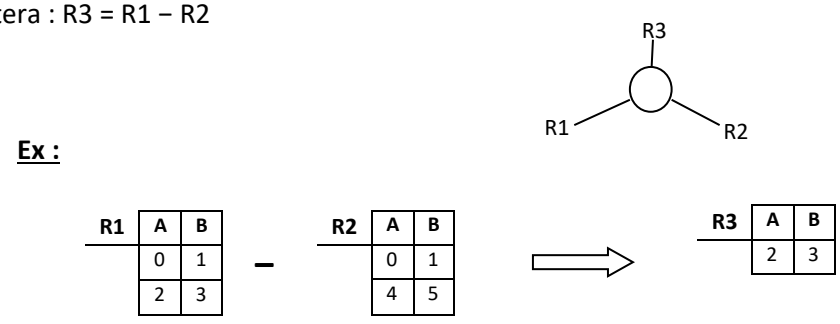
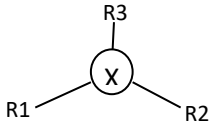


Figure 21. Exemple opérateur Différence

3.3.4 Produit cartésien

Le produit cartésien entre deux relations R1 et R2, est une nouvelle relation R3 contenant l'ensemble de tous les tuples obtenus par concaténation de chaque tuple de R1 avec chaque tuple de R2.

On notera : R3 = R1 X R2



Ex :

R1	A	B
	a1	b1
	a1	b2
	a3	b5

X

R2	C	D	E
	c2	d1	e3
	c3	d4	e1

⇒

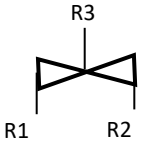
R3	A	B	C	D	E
	a1	b1	c2	d1	e3
	a1	b1	c3	d4	e1
	a1	b2	c2	d1	e3
	a1	b2	c3	d4	e1
	a3	b5	c2	d1	e3
	a3	b5	c3	d4	e1

Figure 22. Exemple opérateur Produit cartésien

3.3.5 Jointure

La jointure de deux relations R1 et R2 est une relation R3 dont les tuples sont obtenus en concaténant les tuples de R1 avec ceux de R2 et en ne gardant que ceux qui vérifient la condition de liaison.

On notera : $R3 = R1 \bowtie_{\text{Condition}} R2$



La condition doit être du type : <attribut1> <operateur> <attribut2>

Où attribut1∈R1, attribut2∈R2 et operateur de comparaison (égalité ou inégalité).

La jointure de deux relations R1 et R2 est le produit cartésien des deux relations suivies d'une restriction. Il existe plusieurs types de jointure :

i. Jointure naturelle

La jointure naturelle est une équijointure des relations R1 et R2 qui ont un attribut commun (appelons X). Le schéma du résultat est similaire au schéma du produit cartésien, modulo que l'attribut X n'apparaisse qu'une fois.

Au niveau des lignes, on combine les lignes de R1 avec les lignes de R2 qui ont même valeur pour l'attribut X, dont l'opérateur de comparaison de la condition est l'égalité.

Ex : $R3 = R1 \bowtie_{R1.A=R2.A} R2$

R1	A	B	C
	a1	b1	c1
	a2	b2	c2

⋈

R2	A	D
	a2	d2
	a2	d3
	a3	d4

⇒

R3	A	B	C	D
	a2	b2	c2	d2
	a2	b2	c2	d3

Figure 23. Exemple opérateur Jointure naturelle

ii. Jointure thêta

Le thêta jointure définit une relation qui contient les tuples qui satisfont le prédicat P du produit cartésien de R1 et R2. Le prédicat P est de la forme R1.ai θ R2.bj où θ est l'un des opérateurs de comparaison (<,≤,>,≥,=,≠).

Ex : $R3 = R1 \underset{R1.A>R2.C}{\bowtie} R2$

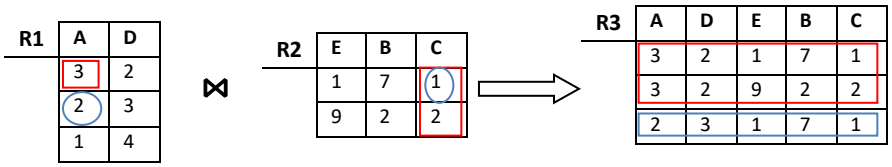


Figure 24. Exemple opérateur Jointure thêta

iii. Jointure externe

La jointure externe définit une relation qui contient toutes les combinaisons significatives entre tuples de deux relations R1 et R2. La jointure externe met en évidence les tuples d'une relation (gauche ou droite), les tuples n'ayant pas de correspondance sont concaténés a des valeurs NULL pour les attributs de l'autre table.

Ex : $R3 = \text{Left } R1 \underset{R1.A=R2.A}{\bowtie} R2$

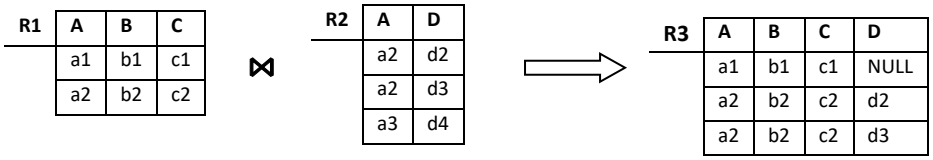


Figure 25. Exemple opérateur Jointure externe

3.3.6 Division

Soit deux relations R1 (A1, A2, … , An, B1, B2, … , Bm) et R2 (B1, B2, … , Bm). Si le schéma de R2 est un sous-schéma de R1, La division de R1 par R2 est une relation R3 dont :

- Le schéma est le sous-schéma complémentaire de R2 par rapport à R1
- Un n-uplet (a1, a2, … , an) appartient à R3 si (a1, a2, … , an, b1, b2, … , bm) appartient à R1 pour tous (b1, b2, … , bm) ∈ R2.

On notera : R3 = R1 ÷ R2

Ex :

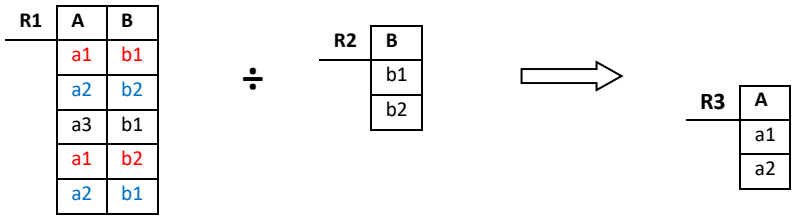


Figure 26. Exemple opérateur Division

3.3.7 Traduction en SQL

a. Opérateurs d'union, d'intersection et de différence

- **Union :**
```
SELECT * FROM table1
UNION
SELECT * FROM table2
```

- **Intersection :**
```
SELECT * FROM table1
INTERSECT
SELECT * FROM table2
```

- **Différence :**
```
SELECT * FROM table1
EXCEPT
SELECT * FROM table2
```

b. Produit cartésien (sans jointure)

Dans le langage SQL, la commande CROSS JOIN est un type de jointure sur 2 tables SQL qui permet de retourner le produit cartésien.

Syntaxe :
```
SELECT *
FROM table1
CROSS JOIN table2
```

Méthode alternative pour retourner les mêmes résultats :
```
SELECT *
FROM table1, table2
```

c. Jointure de tables (condition de jointure)

- **Jointure naturelle :**
```
SELECT *
FROM table1
INNER JOIN table2 ON table1.id = table2.fk_id
```

La jointure SQL peut aussi être écrite de la façon suivante :
```
SELECT *
FROM table1, table2
WHERE table1.id = table2.fk_id
```

- **Jointure externe :**
```
SELECT *
FROM table1
[LEFT/RIGHT/FULL] JOIN table2
ON table1.id = table2.fk_id
```

LEFT : permet de lister tous les résultats de la table de gauche (left = gauche) même s'il n'y a pas de correspondance dans la deuxième table.
Right : permet de retourner tous les enregistrements de la table de droite (right = droite) même s'il n'y a pas de correspondance avec la table de gauche (les colonnes de la table de gauche auront NULL pour valeur.)
FULL : permet de combiner les résultats des 2 tables, les associer entre eux grâce à une condition et remplir avec des valeurs NULL si la condition n'est pas respectée.

d. Fonctions d'agrégat

Considérons la relation Ventes (id, client, tarif, date), il existe plusieurs fonctions qui peuvent être utilisées pour manipuler plusieurs enregistrements, il s'agit des fonctions d'agrégations statistiques, les principales sont les suivantes :

- AVG() pour calculer la moyenne d'un ensemble de valeur, par exemple AVG(tarif) permet de connaître le prix du panier moyen de chaque client.
- COUNT() pour compter le nombre de lignes concernées , par exemple COUNT(*) permet de savoir combien d'achats a été effectué par chaque client.
- MAX() pour récupérer la plus haute valeur, par exemple MAX(tarif) pour savoir l'achat le plus cher.
- MIN() pour récupérer la plus petite valeur, MIN(date) par exemple pour connaître la date du premier achat d'un client.
- SUM() pour calculer la somme de plusieurs lignes, SUM(tarif) permet par exemple de connaître le total de tous les achats d'un client.

Ces fonctions se révèlent rapidement indispensable pour travailler sur des données.

e. Clause GROUP BY et HAVING

i. GROUP BY

La commande GROUP BY est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction d'agrégat sur un groupe de résultat.

Syntaxe :
```
SELECT colonne1, fonction(colonne2)
FROM table
GROUP BY colonne1
```

Ex: Ventes (id, client, tarif, date)

Ventes	id	client	tarif	date
	1	Ali	602	2018-10-23
	2	Omar	470	2018-10-27
	3	Fatima	180	2018-11-05
	4	Fatima	200	2018-11-14
	5	Ali	160	2018-12-03

Figure 27. Exemple GROUP BY

Sur la table ventes d'un magasin, il est possible de regrouper les ventes par clients identiques et d'obtenir le coût total des achats pour chaque client :

```
SELECT client, SUM(tarif)
FROM ventes
GROUP BY client
```

client	SUM(tarif)
Ali	762
Omar	470
Fatima	380

ii. HAVING

La condition HAVING en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que SUM(), COUNT(), AVG(), MIN() ou MAX().

Syntaxe :
```
SELECT colonne1, SUM(colonne2)
FROM nom_table
GROUP BY colonne1
HAVING fonction(colonne2) operateur valeur
```

Ex: Ventes (id, client, tarif, date)
Sur la table ventes d'un magasin, il est possible de récupérer la liste des clients qui ont commandé plus de 400 DA :

```
SELECT client, SUM(tarif)
FROM achat
GROUP BY client
HAVING SUM(tarif) > 400
```

client	SUM(tarif)
Ali	762
Omar	470

3.4 Exemples de requête en SQL et Algèbre relationnelle

Soit la BD :
Pilote (n°pil, nom, adresse, salaire)
Vol (n°vol, n°pil, n°av, VD, VA, HD , HA)
Avion (n°av, nom av, capacité, localisation)

1) Donner les noms des avions de capacité>100

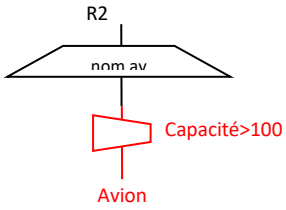
En AR :

$R2 = \pi_{nom\ av}\ R1$

$R2 = \pi_{nom\ av}\ \sigma_{(capacité>100)}\ Avion$

$R1 \leftarrow \sigma_{(capacité>100)}\ Avion$

$R2 \leftarrow \pi_{nom\ av}\ R1$



En SQL :

```
Select nom av
From Avion
Where capacité>100 ;
```

2) Donner les n°vol en partance d'Oran

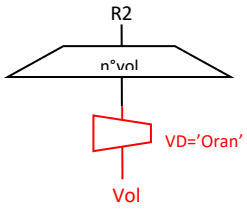
En AR :

$R2 = \pi_{n°vol}\ R1$

$R2 = \pi_{n°vol}\ \sigma_{(VD='Oran')}\ Vol$

$R1 \leftarrow \sigma_{(VD='Oran')}\ Vol$

$R2 \leftarrow \pi_{n°vol}\ R1$



En SQL :

```
Select n°vol
From Vol
Where VD='Oran';
```

3) Donner les noms des pilotes qui assurent des vols en partance d'Oran

En AR :

$R3 = \pi_{nom}\ R2$

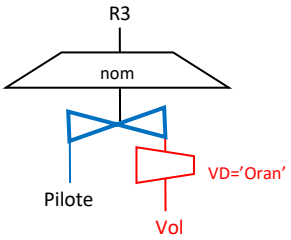
$R3 = \pi_{nom}\ (Pilote \bowtie R1)$

$R3 = \pi_{nom}\ (Pilote \bowtie \sigma_{(VD='Oran')}\ Vol)$

$R1 \leftarrow \sigma_{(VD='Oran')}\ Vol$

$R2 \leftarrow Pilote \bowtie_{Pilote.n°pil=Vol.n°pil} R1$

$R3 \leftarrow \pi_{nom}\ R2$



En SQL :

```
Select nom
From Pilote, Vol
Where Pilote.n°pil=Vol.n°pil and VD='Oran';
```

Bibliographie

- Bases de données. Georges Gardarin. 5ème édition 2003
- SQL Les fondamentaux du langage. Eric Godoc et Anne-Christine Bisson. Edition Eni. 2017
- Bases de données : concepts, utilisation et développement. Jean-Luc Hainaut. Édition DUNOD. 2015

Cours Mis en-ligne pour les étudiants moodle

https://elearning.univ-usto.dz/course/view.php?id=175