

Université Louis-Pasteur de Strasbourg
Département d'informatique

Bases de Données

Sylvain BRANDEL

Pôle API
Boulevard Sébastien Brant
67 400 ILLKIRCH
bureau C 126
tél : 03 90 24 45 68
e-mail : brandel@dpt-info.u-strasbg.fr

Notes de cours (DESS CCI)

(Imprimé le 19 juillet 2018)

Table des matières

Chapitre 1 : BD et SGBD	3
I. Introduction.....	3
II. Analyse	3
III. Bases de données.....	6
IV. Système de Gestion des Bases de Données (SGBD)	7
Chapitre 2 : Le modèle relationnel	19
I. Définitions	19
II. Langages d'interrogation.....	20
III. Dépendances fonctionnelles	29
IV. Normalisation de relation.....	34
V. Insertion - suppression	41
VI. SQL.....	42
Chapitre 3 : Le modèle Entités / Associations	45
I. Entité.....	45
II. Association	45
III. Cardinalités et fonctionnalités	46
IV. Attribut.....	48
V. Règles de normalisation	49
VI. Passage du modèle E/A au modèle relationnel	50
VII. Concepts complémentaires}	52
VIII. Exercices de synthèse	55

Chapitre 1 : BD et SGBD

Références :

Bases de Données objet et relationnel
Georges Gardarin
éd. Eyrolles

Introduction aux bases de données, modèle relationnel
Richard Grin (support de cours)
Université de Nice Sophia-Antipolis

I. Introduction

BD, c'est quoi ?

Déf. : Une BD est un ensemble de données modélisant les objets d'une partie du monde réel et servant de support à une application informatique.

→ ensemble :

- structuré
- permettant l'extraction d'informations de manière sélective par plusieurs utilisateurs

Exemple : annuaire (Nom, Prénom, âge, CP, ville)

- d'abord un tableau (fichier) → redondance
- 2^{ème} tableau (CP, ville)
- 3^{ème} tableau (âge) ? → redondance calculée

Deux types d'opérations :

- Manipulation (utilisateur)
- Définition (concepteur, administrateur)

Tout ensemble de données ne justifie pas forcément une BD.

Exemple : CDs audio

- suffisamment de CDs pour nécessiter une indexation informatique
- mais si le seul but est d'imprimer la liste, un éditeur de textes suffit

Déf. : Un SGBD est un logiciel permettant d'interagir avec une BD.

Exemples de SGBD : Access, Oracle. Pas SQL.

II. Analyse

Avant de penser BD, poser des questions, étudier, analyser les besoins.

Exemple : CD audio → informatiser. Pourquoi ? dans quel but ? quand ? comment ?

→ analyse.

Exemple : Ecole. Dialogue avec la directrice.

- pour obtenir des règles de gestion
- pour étudier les formulaires, fiches... existant
- dictionnaire des données

→ MERISE

A. Pourquoi MERISE ?

Problèmes des grosses applications :

- applications non coordonnées :

- redondance
- synonymes
- nombreuses interfaces
- difficultés de MAJ
- applications "touffues" :
 - astuces de programmes, GOTO, optimisations ponctuelles
 - absence de dossier
- absence de concertation → résultats différents des souhaits

B. Principes de base

Méthode globale et intégrée :

- repenser en terme de SAI (Système d'Automatisation de l'Information) et d'organisation
- intégrer les différents utilisateurs

Séparation Données / Traitements : analyses conduites séparément puis confrontation

3 niveaux de modélisation, invariance décroissant (plus le niveau est élevé, et plus le système est stable) :

- niveau conceptuel
- niveau organisationnel
- niveau technique

1) Niveau Conceptuel : "QUOI ?"

Que faire ? avec quelles données ?

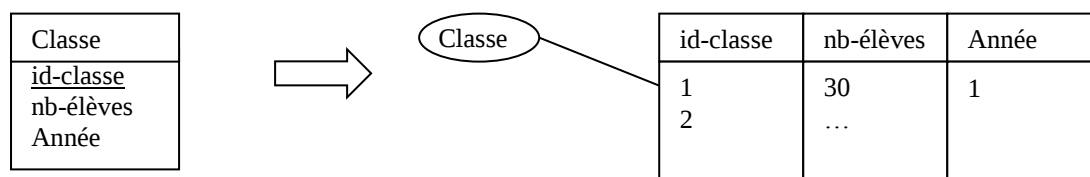
→ MCD (Modèle Conceptuel des Données) et MCT (Modèle Conceptuel des Traitements)

a) MCD

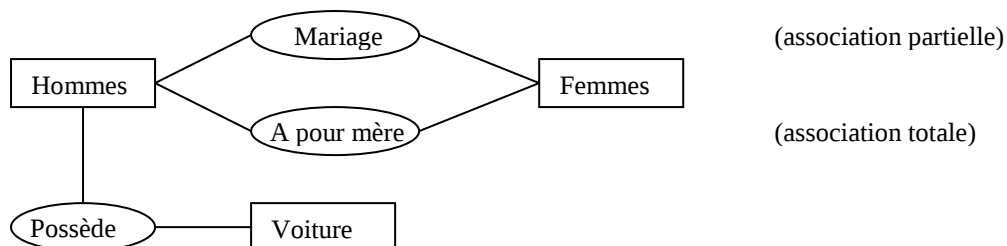
aspect statique du Système d'Information, schéma entité / associations.

Entité : collection d'objets ayant des propriétés analogues et possédant un identifiant.

Exemple : CLASSE < id-classe, nb-élèves, année >



Association : Lien logique entre plusieurs entités.



b) MCT

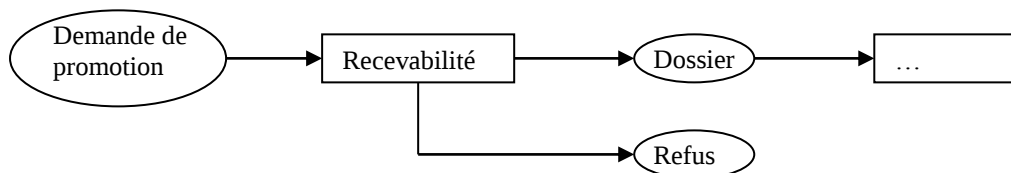
ce qu'il faut faire : QUOI (pas OU, QUAND → organisationnel, pas COMMENT → opérationnel).

Événement : compte-rendu fait au SI que quelque chose s'est produit dans l'univers extérieur ou dans le SI.

Les événements peuvent être :

- externes (par exemple une commande)
- internes (par exemple une facture)

Opération : actions accomplies par le SI en réaction à un (ou des) événement(s).



2) Niveau Organisationnel : "QUI, QUAND, OU ?

→ MLD (Modèle Logique des Données) et MOT (Modèle Organisationnel des Traitements)

a) MLD

"Représentation du MCD en fonction de l'état de l'art technologique" (Tardieu)

= MCD repensé.

b) MOT

Modéliser :

- les notions de temps et de durée
- les postes et leur responsabilités
- les lieux et leurs ressources

Exemples de règles d'organisation :

- à chaque livraison, le magasinier contrôle la marchandise livrée
- l'inventaire est effectué tous les ans les 29 et 30 décembre

3) Niveau Technique : "COMMENT ?"

Intègre les moyens techniques, matériels et logiciels.

→ MPhD (Modèle Physique des Données) et MOpT (Modèle opérationnel des Traitements)

a) MPhD

proposer une solution technique optimale pour l'implantation physique des données et la description de leurs liens.

b) MOpT

intégrer les contraintes techniques dans l'organisation des traitements.

C. Inconvénients de MERISE

Lourdeur :

- produit beaucoup de documents
- lourd pour des petites applications

Par rapport aux étudiants :

- lourdeur sur les exemples pédagogiques
- sentiment de flou : haut niveau d'abstraction, et méthodologie rigoureuse mais non déterministe (possibilité de production de différents algorithmes)

III. Bases de données

BD : Collection de représentation de la réalité sous forme de données inter-reliées :

- cohérente, structurée
- avec une redondance calculée
- centralisée

A. Les précurseurs des BD : les systèmes de fichiers

Dans une entreprise, chaque service avait ses propres fichiers, développés en fonction de ses besoins

→ informations communes, sans que cela soit géré.

→ redondance non gérée.

B. Redondance

Problèmes :

- volume de données important (moins grave actuellement : mémoires de masse importantes)
- temps de saisie et de mise à jour : par exemple, si on dispose de 3 fichiers clients (factures, BL, étiquettes), la saisie d'un client doit être effectuée trois fois, avec des risques d'erreur (fautes de frappe entre deux saisies)
- difficultés de tenir à jour les informations pour "coller" à la réalité
→ manque de cohérence.

C. Cohérence

Etre juste par rapport à une certaine réalité, c-à-d intégrité des données.

La réalité doit être fixée sous forme de règles ou contraintes d'intégrité.

La BD est cohérente si elle respecte les contraintes d'intégrité énoncées (dépendantes des applications).

Les règles définissant les liens entre les données de la base décrivent la réalité.

Exemples de contraintes d'intégrité : BD clients / fournisseurs / produits, avec commandes clients et fournisseurs :

- couleur autorisée pour un produit $\in \{\text{blanc, bleu, noir}\}$
- chaque client, chaque fournisseur, chaque produit doit être identifié de manière unique dans toute la base (→ numérotation explicite)
- pour la gestion des commandes, les clients et produits intervenant dans la transaction doivent exister
- tout client sauvegardé dans la base a passé au moins une commande

Les contraintes sont posées au départ :

- elles définissent la structure de la base
- elles doivent être vérifiées en permanence

D. Centralisation

La BD doit satisfaire une grande variété de demandes de renseignements exprimée par de nombreux utilisateurs.

Les utilisateurs ont des exigences de réponse qui doivent être compatibles avec leurs conditions de travail.

→ BD : entité commune, centrale, à laquelle accèdent tous les utilisateurs.

L'utilisateur doit pouvoir travailler comme il l'a toujours fait, avec sa propre vision de la BD.

IV. Système de Gestion des Bases de Données (SGBD)

Un SGBD est une ensemble de logiciels permettant aux utilisateurs de manipuler efficacement des données dans une grande masse d'information partagée avec d'autres utilisateurs.

Différence avec les systèmes de fichiers : description des données séparée de leur utilisation (description : définition des types par des noms, des formats, des caractéristiques ; utilisation : interrogations, mises à jour).

Le système est en fait un composant de plus bas niveau, englobé dans le SGBD.

A. Historique

4 générations. Le début se situe vers 1965 avec le programme Apollo d'IBM.

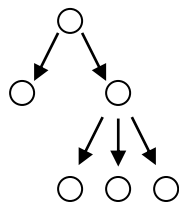
1) 1^{ère} génération : fin des années 60

Exemple : TOTAL, IDMS.

Modèles réseau ou hiérarchique, organisés autour de types d'articles constituant les nœuds d'un graphe, les arcs étant des types de pointeurs.

a) Modèles hiérarchiques

Les données sont représentées sous forme d'une structure arborescente d'enregistrements. Cette structure est conçue avec des pointeurs et détermine les chemins d'accès aux données.



Avantages :

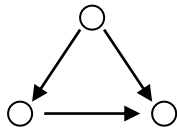
- structure de donnée facile à comprendre
- chemins d'accès fixes : peu de commandes, simplicité des commandes

Inconvénients :

- organisation non naturelle
- contrôle de cohérence délicat
- coût élevé de la recherche, dû à l'unicité du chemin d'accès

b) Modèles réseau

La structure peut être visualisée sous forme d'un graphe quelconque. Comme pour le modèle hiérarchique, la structure est conçue avec des pointeurs et détermine le chemin d'accès aux données.



Avantages :

- moins restrictif
- représentation naturelle des liaisons
- moins de redondances

Inconvénients :

- pas d'indépendance vis-à-vis des stratégies d'accès

Dans ces deux modèles, les programmes ne sont pas indépendants de la structure logique de la base et du chemin d'accès aux données : ils doivent décrire comment retrouver les données (on parle de *navigation* dans la base) et si, par exemple, on enlève un index, tous les programmes qui l'utilisaient doivent être réécrits. De plus, le langage de travail est complexe.

2) 2^{ème} génération : fin des années 70

Exemple : ORACLE, SYBASE, INFORMIX, DBASE, SQL SERVER.

Modèle relationnel.

Point de départ : théorie des ensembles en mathématiques.

Les données sont présentées aux utilisateurs sous forme de relations entre domaines de valeurs, simplement représentées par des tables, et plus les pointeurs qui figeaient la structure de la base.

Les recherches et mises à jour sont effectuées à l'aide d'un langage non procédural standardisé :

SQL (Structured Query Language), permettant de traduire les requêtes du langage courant.

Les SGBD de la 2^{ème} génération représentent l'essentiel du marché actuel.

Avantages :

- simplicité de la structure
- indépendance totale entre niveaux logiques et physiques

Inconvénients :

- langage pas adapté aux données complexes (les images par exemple)
- sémantique pauvre
- pas de point de vue opérationnel des données
- déductions limitées

L'inventeur du modèle relationnel est Codd (IBM). Il a énoncé 12 règles que doivent vérifier les SGBD pour être relationnels (actuellement aucun SGBD ne les respecte toutes, mais certains tentent de s'approcher de cette « perfection » relationnelle) :

- **règle 1 :** toutes les informations sur les données sont représentées au niveau logique et non physique (pas besoin d'informations sur la façon dont sont enregistrées physiquement les données)
- **règle 2 :** les données sont accessibles uniquement par la combinaison du nom de la table, de la clé primaire et du nom de la colonne (pas de chemin à donner)
- **règle 3 :** une valeur spéciale doit représenter l'absence de valeur (NULL)
- **règle 4 :** la description de la base de données doit être accessible comme les données ordinaires (un dictionnaire des données est enregistré dans la base)
- **règle 5 :** un langage doit permettre de définir les données, définir des vues (visions particulières de la base, enregistrées comme des relations), manipuler les données, définir des contraintes d'intégrité, des autorisations de gérer des transactions
- **règle 6 :** on peut faire des mises à jour par les vues lorsque c'est logiquement possible
- **règle 7 :** le langage doit comporter des ordres effectuant l'insertion, la mise à jour et la suppression des données (un seul ordre pour effectuer chacune de ces fonctions)
- **règle 8 :** indépendance des programmes vis-à-vis de l'implantation physique des données
- **règle 9 :** indépendance des programmes vis-à-vis de l'implantation logique des données (si les informations manipulées par les programmes n'ont pas été modifiées ou supprimées)

- **règle 10** : les contraintes d'intégrité doivent pouvoir être définies dans le langage relationnel et enregistrées dans le dictionnaire des données
- **règle 11** : indépendance vis-à-vis de la répartition des données sur divers sites
- **règle 12** : on ne peut jamais contourner les contraintes (d'intégrité ou de sécurité) imposées par le langage du SGBD en utilisant un langage de plus bas niveau (par exemple le C)

3) 3^{ème} génération : fin des années 80

Les structures de données complexes sont éclatées par le modèle relationnel et la reconstruction de la structure nécessite des opérations (jointures) coûteuses en performances.

→ Extensions objet des systèmes relationnels.

Exemple : ORACLE 8, DB2 Universal Database, Informix Universal Server.

Structuration conjointe des données et des programmes en types : les données sont enregistrées avec les procédures qui permettent de les manipuler.

Possibilités de définir des sous-types par héritage.

Règles actives → répercussions des MAJ d'un objet sur d'autres objets dépendants.

4) 4^{ème} génération : années 90 et actuellement

Plus extension de la 3^{ème} génération que vraiment une nouvelle génération.

Permet de mieux supporter Internet et le Web, les informations mal structurées, les objets multimédia, l'aide à la décision et l'extraction de connaissances.

Mais cela reste un système relationnel → domaine (actif) de recherche Intelligence Artificielle (IA).

Il n'existe pas encore de norme, et ces SGBD sont encore utilisés pour des usages bien spécifiques : le manque de support et d'uniformité dans les implantations des différents SGBD du marché conduisent à les utiliser avec prudence.

B. Objectifs des SGBD

Un SGBD est un ensemble de logiciels qui fournit un environnement pour :

- décrire des données
- mémoriser des données
- manipuler des données (ajouter, modifier, supprimer)
- traiter des données en les restituant éventuellement sous une forme différente de leur saisie (selectionner, trier, calculer...)
- définir des contraintes d'intégrité sur les données (contraintes de domaines, d'existence...)
- définir des protections d'accès (mots de passe, autorisations...)
- résoudre les problèmes d'accès multiples aux données (blocages, interblocages...)
- prévoir des procédures de reprise en cas d'accident (sauvegardes, journaux...)

Un SGBD doit en outre permettre d'écrire des applications indépendantes de l'implantation physique des données (codage, ordre dans lequel les données sont enregistrées...).

Objectif : assurer

- la sécurité
- la confidentialité
- l'intégrité
- de bonnes performances

des données alors que plusieurs utilisateurs peuvent accéder simultanément à la BD.

1) Sécurité

Une perte d'informations peut s'avérer fatale → la sécurité est primordiale.

a) Sécurité de fonctionnement

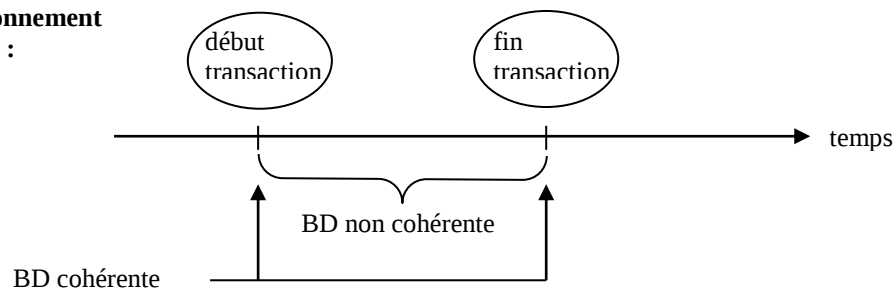
Suite à un incident matériel ou logiciel, il convient de :

- redémarrer le système
- remettre la BD en état cohérent

→ archivage : copie de sauvegarde sur un autre support (CD, DAT, disquette...).

→ journalisation : un journal mémorise toutes les transactions effectuées sur la BD. En cas de panne de courant, le système lit le journal.

Fonctionnement normal :



Durant la transaction (ensemble de modifications de la base qui forme un tout indivisible), des opérations sont effectuées sur la BD → BD non cohérente temporairement.

→ risque, si un autre utilisateur accède à la BD à ce moment.

La transaction doit être terminée ou non exécutée.

Exemple de transaction : transfert d'une somme d'argent entre deux comptes d'un client d'une banque. Elle comporte deux ordres : débiter le premier compte, puis créditer le second. Si un problème empêche le crédit, le débit doit être annulé.

Une transaction est terminée :

- soit par une validation qui entérine les modifications
- soit par une annulation qui remet la base dans son état initial

Deux transactions ne peuvent se chevaucher.

L'utilisateur peut à tout moment valider la transaction en cours (COMMIT en SQL) → les modifications deviennent alors définitives et visibles à l'ensemble des utilisateurs.

Tant que la transaction n'est pas validée, les modifications qu'elle contient n'apparaissent qu'à l'utilisateur qui l'exécute. Tous les autres utilisateurs voient la base dans l'état où elle était avant le début de la transaction.

L'utilisateur peut annuler la transaction en cours (ROLLBACK en SQL) → les modifications effectuées depuis le début de la transaction sont annulées.

b) Gestion de la concurrence d'accès

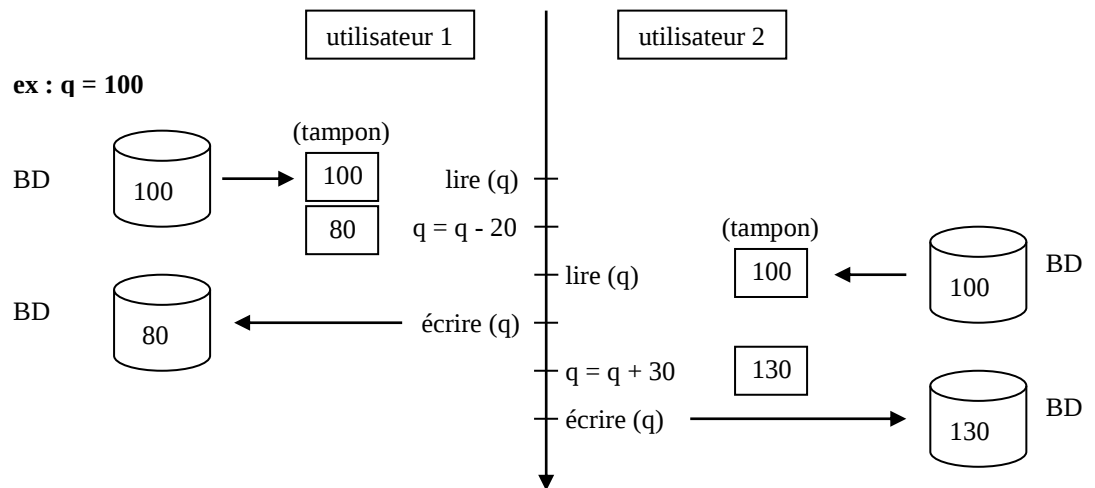
Lorsque plusieurs personnes travaillent simultanément sur la même donnée, il faut éviter que le travail de l'une écrase celui de l'autre.

Exemple : éditer un même fichier partagé.

Mises à jour perdues. Par exemple, 2 personnes veulent modifier une quantité q en stock :

Transaction :

- lire (q)
- modifier (q) (valeur stockée dans un buffer)
- écrire (q) (dans la BD)



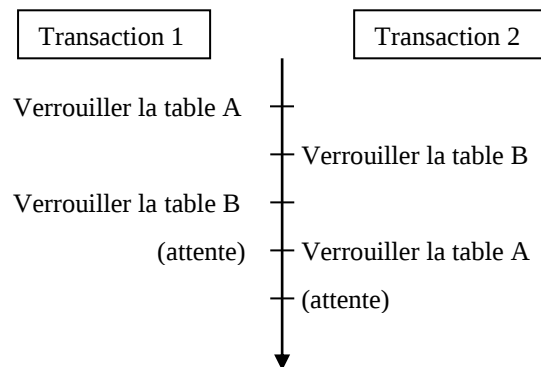
Ici, le travail de l'utilisateur 1 est écrasé.

→ Mécanisme des verrous :

- interdit l'accès à une donnée (table ou partie de table) tant que celle-ci est utilisée
- verrou pendant un temps minimal → généralement transparent (tous les utilisateurs ont l'impression de travailler seuls sur la donnée)

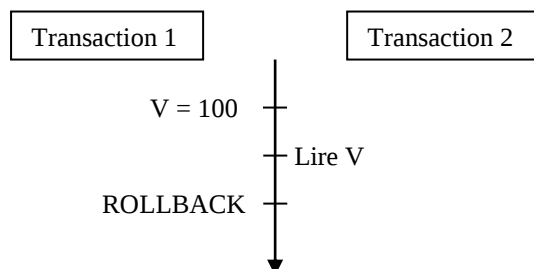
Mais cela peut conduire à un interblocage de processus (*deadlock*) :

Exemple :



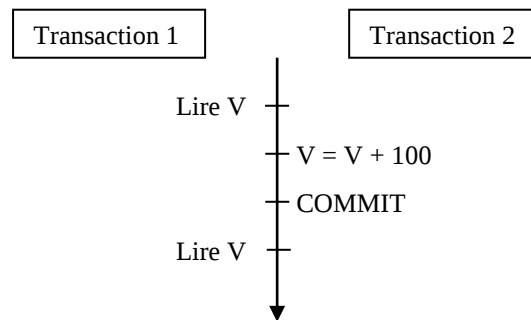
Ce type d'interblocage ne peut pas arriver si tous les utilisateurs bloquent les tables dans le même ordre.

Lecture inconsistante ou lecture impropre. Une transaction T2 lit une valeur donnée par une autre transaction T1. Ensuite la transaction T1 annule son affectation de V → la valeur lue par T2 est fausse :



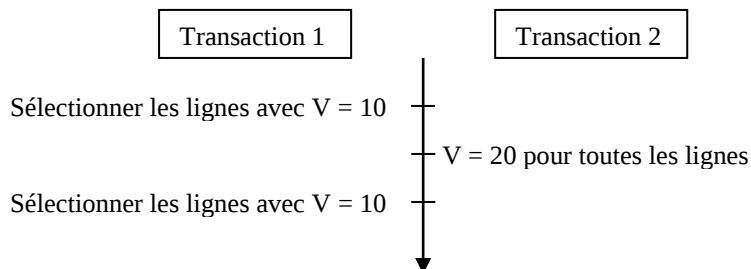
Ce cas ne peut pas arriver si les modifications effectuées par une transaction ne sont visibles par les autres qu'après validation (COMMIT) de la transaction.

Lecture non répétitive, ou non reproductible, ou incohérente. Une transaction lit deux fois une même valeur et ne trouve pas la même valeur les deux fois.



Pour éviter ce cas, T1 devra bloquer les données qu'il veut modifier suffisamment longtemps pour empêcher les autres transactions de les modifier.

Lignes fantômes. Une sélection de lignes récupère des lignes qui sont modifiées par une autre transaction et ne vérifient plus le critère de la sélection. La même sélection lancée une seconde fois ne retrouve donc plus ces lignes.



Là encore la solution est le blocage explicite des lignes en question.

Traitement des accès concurrents par le SGBD. Les SGBD gèrent automatiquement les accès concurrents de plusieurs utilisateurs sur les mêmes lignes des tables.

Dans Oracle, une donnée est en cours de modification → les autres utilisateur ::

- peuvent lire la donnée telles qu'elles étaient avant cette modification (jamais de délai d'attente pour la lecture)
- sont bloqués automatiquement s'ils veulent modifier la donnée

Oracle assure une « lecture consistante » des données pendant l'exécution d'un ordre SQL : par exemple, un ordre SELECT ou UPDATE va travailler sur les lignes telles qu'elles étaient au moment du début de l'exécution de la commande, même si entre-temps un autre utilisateur a confirmé (COMMIT) des modifications sur certaines de ces lignes.

2) Confidentialité

a) Contrôle de l'accès à la base

Gestion des droits d'accès des différents utilisateurs (mots de passe).

Gestion des « privilèges » : les utilisateurs n'ont pas tous les mêmes droits.

b) Contrôle de l'accès aux données

Les données d'une table appartiennent au créateur de la table, qui peut donner aux autres utilisateurs des droits sur cette table (par exemple consulter, ajouter...)

Les « vues » fournissent également des restrictions d'accès aux données. Une vue est une table virtuelle constituée de colonnes qui appartiennent à des tables réelles. On peut donner le droit d'accès à une vue sans donner le droit d'accès aux tables sous-jacentes.

On peut aussi donner des droits sur une partie seulement des lignes ou des colonnes d'une table.

3) Intégrité

Les opérations doivent respecter les contraintes d'intégrité implicitement :

- définition de la clé primaire d'une table
- clés étrangères dont les valeurs doivent exister dans une autre table
- unicité des valeurs d'une rubrique
- fourchette pour les valeurs acceptables
- ...

Les commandes qui ne respectent pas ces contraintes sont automatiquement annulées (toute la commande est annulée si une seule des lignes ne respecte pas une contrainte d'intégrité).

Exemple : BD client / fournisseur / produit

- interdire l'insertion d'un produit dont le numéro figure déjà dans la base
- interdire l'insertion d'une nouvelle commande avec un client inconnu
→ il faut d'abord insérer le client, puis la commande
- la suppression d'une commande peut provoquer la suppression d'un client (s'il n'avait qu'une commande)

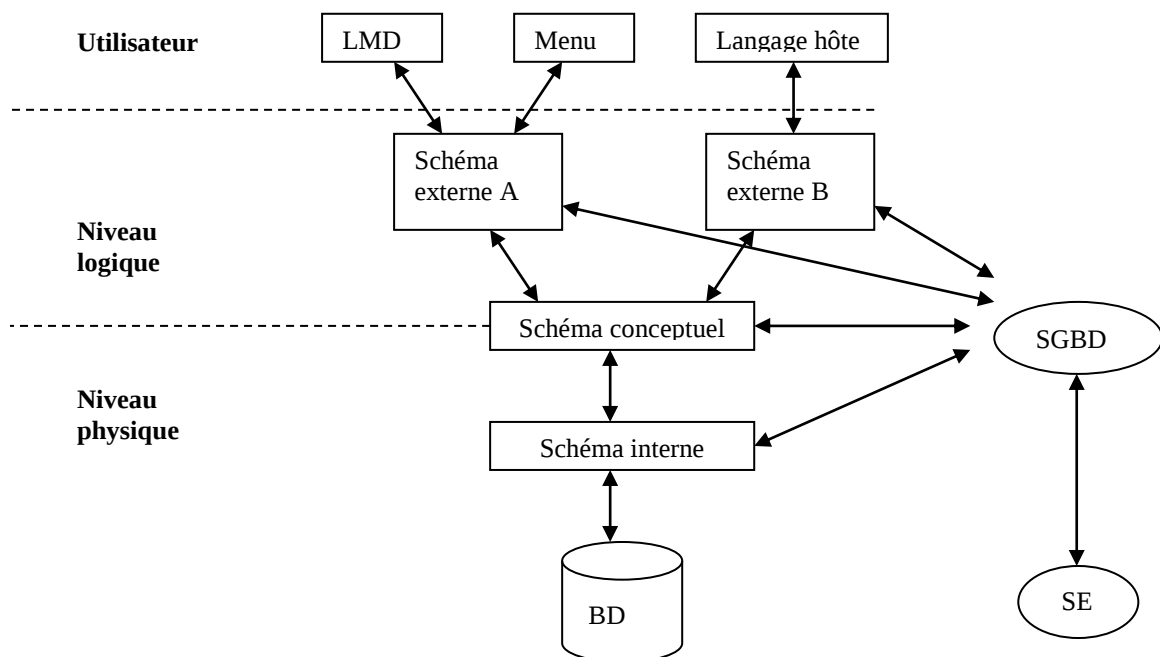
4) Performances

Les langages d'utilisation des SGBD relationnels ne font pas référence au cheminement à suivre pour accéder aux données → risque de performances dégradées (données non contiguës, isolées...).

Solutions :

- index sur une rubrique → accélérer l'accès aux lignes qui ont une certaine valeur ou accélérer le tri des lignes dans l'ordre de rangement de l'index
- clusters : rangement physique de plusieurs tables dans un même espace → accélérer les jointures sur ces tables en partageant des colonnes entre plusieurs tables

C. Architecture des SGBD



3 schémas (norme ANSI/SPARC) :

- conceptuel
- externe
- interne

Ce cours concerne particulièrement le niveau conceptuel.

3 types d'utilisateurs :

- administrateur
- programmeur
- utilisateur final

2 langages :

- LMD
- LDD

1) Schémas

a) Schéma conceptuel

Niveau central, vision globale de la base. Un seul schéma conceptuel. Description des données et des CI qui y sont liées.

Différentes approches :

- liaison : schémas externes / schéma conceptuel
- ou schéma externe = dérivation du schéma conceptuel
- ou schéma conceptuel = synthèse des schémas externes

→ deux utilisateurs peuvent appeler une même variable sous deux noms différents, ou bien appeler deux variables différentes sous un même nom.

Exemple :

BUVEUR(Nom, Prénom, Adresse)	(objet)
VIN(Cru, Millésime, Quantité)	(objet)
ABUS(Buveur, Vin, Date, Quantité)	(association)

b) Schémas externes

Vision des données d'un utilisateur ou d'un groupe d'utilisateurs → description d'une partie Seulement des données → vision parcellaire, incomplète.

→ Sécurité et confidentialité des données.

Il existe autant de visions externes que d'utilisateurs.

Mais les visions des différents utilisateurs vont se recouper (informations communes).

Exemple :

Schéma externe 1 :

buveur-de-vin :

Nom Prénom NbAbus

Schéma externe 2 :

identité-buveur :

Nom Prénom

vins-consommés :

Cru Quantité

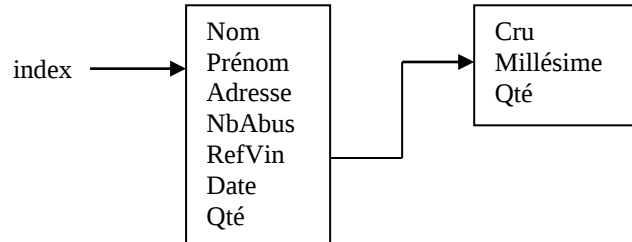
c) Schéma interne

Description de la manière dont les données sont stockées sur les organes physiques.

Dans un SGBD, l'organisation physique des données est effectuée par le système d'exploitation → ignoré par l'utilisateur.

Il peut y avoir plusieurs modèles physiques pour un même modèle logique.

Exemple :



d) Indépendances

Indépendance logique si le schéma conceptuel peut être modifié ou enrichi sans que cela affecte les schémas externes des utilisateurs et leurs programmes d'application.

Indépendance physique si on peut modifier le schéma interne sans affecter le schéma conceptuel. Cela permet de ne pas être lié au matériel ou au système d'exploitation (seul le niveau interne en dépend).

2) Utilisateurs

a) Administrateur

Personne(s) responsable(s) de tout ce qui touche à la BD :

- bon fonctionnement
- création de la BD
- déclaration d'un utilisateur
- droits d'accès
- modification des schémas externes
- modification des chemins d'accès aux données
- maintien des performances d'accès aux données
- sauvegardes et reprises après pannes
- ...

b) Programmeur

Développeur d'applications utilisant la BD. Il a le droit de créer de nouvelles tables et les structures associées (vues, index, clusters...). Il définit les droits qui seront accordés aux utilisateurs des applications qu'il développe.

c) Utilisateur final

N'a accès qu'aux données qui lui sont utiles. Il a certains droits accordés par l'administrateur : consultation, ajout, modification ; suppression. Généralement il n'a pas le droit de création, de modification ou de suppression de tables.

3) Langages

a) Langage de Définition des Données (LDD)

Utilisé par l'administrateur de la base, le LDD permet de définir :

- le schéma conceptuel et les schémas externes
- les données, liens, contraintes d'intégrité

Utilisé lors de la conception de la base et la modification des schémas.

b) Langage de Manipulation des Données (LMD)

Interrogation et MAJ de la BD.

Les instructions peuvent être :

- interactives (outils en ligne, interfaces avec des outils de bureautique ou avec le web)
- insérées dans un langage de programmation = langage hôte de 3^{ème} génération (C → Pro*C) ou de 4^{ème} génération (langages conçus spécifiquement pour développer des applications qui utilisent des BD)
- sous forme de menus (forms...), de formulaires, d'états imprimés...

D. Exemple de fonctionnement d'un SGBD

Le programme d'application, accédant à la BD via le schéma externe B, demande la lecture d'une donnée de la base.

1. Demande transmise au SGBD.
2. le SGBD analyse la demande :
 - consultation du schéma externe pour vérifier les droits d'accès à la donnée ; lecture des caractéristiques de la donnée à partir du schéma externe B ;
 - consultation du schéma conceptuel → type logique de la donnée à extraire ;
 - consultation du schéma physique → enregistrement physique à lire.
3. Le SGBD transfère l'ordre de lecture au Système d'Exploitation (SE).
4. Le SE reçoit l'ordre, l'analyse, puis lance un ordre de lecture au contrôleur des unités périphériques.
5. Les données sont placées dans un tampon.
6. Le SGBD sélectionne, dans le tampon, les données effectivement demandées et les transmet au tampon du programme d'application.
7. Le SGBD peut prévenir le programme d'application en cas de déroulement anormal.
8. Le programme d'application dispose de la donnée demandée.

E. Résistance aux pannes

1) Sauvegardes

→ Réparer les dommages créés par des pannes :

- le SGBD enregistre automatiquement toutes les instructions exécutées sur la base dans des fichiers de « log » → fichiers utilisés lors du redémarrage après la panne
- l'administrateur sauvegarde régulièrement les BD et les fichiers utilisés par le SGBD et enregistre les fichiers de log sur des médias stables (bandes, CD) → ralentissement du système mais plus grande fiabilité en cas de panne

2) Types de pannes

- origine logicielle (SGBD ou applications qui accèdent aux données)
- origine matérielle (le plus souvent les disques, mais aussi pannes de courant...)
- dues au réseau en cas de BD distribuées

Le traitement est différent selon que la panne a laissé la base dans un état cohérent (interruption logicielle) ou si elle a été endommagée et ne peut plus être utilisée (crash disque)

3) Reprise après accident

a) Fichiers non endommagés

Utilisation des fichiers de log : les modifications validées (COMMIT) mais non enregistrées réellement (pour des raisons d'écriture asynchrone) sont ré exécutées ; les transactions annulées (ROLLBACK) ou non confirmées (COMMIT) avant la panne sont annulées.

b) Fichiers endommagés

L'administrateur doit recharger la dernière sauvegarde.

Si toutes les modifications ont été archivées (fichiers de log), il peut relancer leur exécution jusqu'à la panne, ou tout au moins jusqu'à la dernière archive valide. Sinon, il faudra relancer à la main toutes les transactions exécutées entre la dernière sauvegarde et la panne.

Les fichiers de log ne servent pas seulement à la réparation en cas de panne, mais sont utilisés pour des raisons de performances : lors d'un COMMIT, les données ne sont pas directement écrites mais leur modification est inscrite dans les fichiers de log → écriture séquentielle plus efficace.

F. Implantation d'un SGBD

Ce paragraphe décrit les solution retenues dans l'implantation du SGBD Oracle (versions 7 et 8i).

1) Fichiers de la base

Tous les SGBD gèrent eux-mêmes l'enregistrement des données sur les mémoires auxiliaires (disques le plus souvent). Les performances sont ainsi meilleures que s'ils utilisaient simplement le système d'exploitation et son système de fichiers. En général, ils se réservent de très gros fichiers du système d'exploitation et ils gèrent eux-mêmes cet espace pour allouer de la place aux tables ,index et autres objets manipulés.

Un autre avantage est la plus grande facilité pour le portage sur des systèmes d'exploitation différents.

2) Processus clients – serveurs

L'utilisation d'un SGBD s'appuie sur le mode client – serveur : les applications des utilisateurs qui accèdent à la base sont clients de processus serveurs lancés en arrière plan par le SGBD.

Les applications clientes n'ont pas d'accès direct aux données de la base. Elles transmettent leurs requêtes (SQL pour Oracle et la plupart des SGBD relationnels) aux processus serveurs du SGBD. Un processus serveur peut prendre en charge l'interface entre la base et un ou plusieurs processus clients. Il y a en général plusieurs processus serveurs qui fonctionnent en même temps.

Un processus serveur exécute les requêtes des clients dont il a la charge et en particulier

- transmet aux clients les résultats des requêtes de consultation de la base (les données seront lues en mémoire dans les buffers de la base si elles y sont déjà),
- dépose dans les buffers de la base (en mémoire) les données ajoutées ou modifiées par les requêtes de manipulation des données.

3) Ecriture des données dans la base

Un seul processus (*Data Base Writer*, DBWR) a la charge d'enregistrer dans les fichiers de la base les données déposées dans les buffers par les processus serveurs. Le fait qu'un seul processus soit chargé de cet enregistrement facilite le maintien de l'intégrité des données.

Pour des raisons de performances cette écriture se fait de manière asynchrone par rapport aux actions des serveurs. Il y a, par exemple, écriture quand les buffers sont pleins. L'écriture des différentes entrées des buffers se fait par l'algorithme LRU (*Least Recently Used*) et ne tient pas compte des COMMIT : les entrées qui sont enlevées des buffers et enregistrées dans la base sont celles qui ont été le moins récemment utilisées. Une entrée très utilisée peut donc n'être enregistrée (physiquement sur le disque) dans la base que longtemps après son écriture dans les buffers de la base.

4) Fichiers log

Parallèlement à l'enregistrement des données, les processus serveurs enregistrent toutes les actions effectuées sur les tables de la base (et dans les segments de *rollback*), dans des fichiers indépendants des fichiers de la base, appelés fichiers *log* (*redo log* dans Oracle).

Comme pour les données de la base, les données enregistrées dans les fichiers *redo log* sont d'abord déposées en mémoire dans des buffers et enregistrées dans les fichiers par un seul processus (*Log Writer* dans Oracle). Ces enregistrements ont lieu régulièrement, par exemple quand les buffers sont pleins, et au moins dès qu'il y a eu un COMMIT.

L'enregistrement d'un COMMIT dans les fichiers *redo log* est l'événement de référence pour la confirmation d'une transaction. Chaque COMMIT déclenche une écriture immédiate sur le disque des entrées du buffer des fichiers *redo log* correspondant à la transaction validée. Cette écriture est rapide car séquentielle alors que l'écriture des modifications dans la base est même plus lente.

Lorsque les fichiers *redo log* ont enregistré un COMMIT et qu'une panne survient avant que les données correspondant à la transaction de la base ne soient écrites dans la base, le SGBD effectue l'enregistrement physique des données après la reprise. Si par contre c'est un ROLLBACK qui a été enregistré et si des modifications ont déjà été enregistrées (par exemple parce que les buffers des fichiers *redo log* étaient pleins), les données sont remises à leur état précédant la transaction après le redémarrage, à l'aide des segments de rollback.

5) Segments de rollback

Quand des données sont modifiées dans une base Oracle, toutes les informations qui permettraient de retrouver les anciennes données (essentiellement les valeurs anciennes des données modifiées) sont enregistrées dans la base dans ce des *segments de rollback*.

Ces segments sont utilisés :

- pour annuler une transaction après une instruction ROLLBACK
- tant qu'une transaction n'est pas confirmée par un COMMIT, les modifications qu'elle apporte aux données ne doivent pas apparaître aux autres transactions (principe de *lecture consistante*).

6) L'optimiseur de requêtes SQL

SQL est un langage non procédural. Par exemple, une instruction SELECT décrit les données recherchées sans indiquer la manière d'aller les chercher dans la base.

L'optimiseur est le module du SGBD qui va concevoir un plan pour aller rechercher les données de la manière la plus efficace possible. Il utilisera généralement pour cela les index.

L'optimiseur utilise des règles de simplification et d'optimisation implantées par les concepteurs du SGBD, et les données statistiques sur les tables (par exemple, nombre de lignes et nombre de valeurs différentes dans les colonnes).

Chapitre 2 : Le modèle relationnel

I. Définitions

A. Relation

Une *relation* R est sous-ensemble du produit cartésien de n ensembles $D_1, D_2, \dots, D_n$ appelés *domaines* ; n est appelé le *degré* de la relation. Une relation est constituée de n -uplets de la forme $(d_1, d_2, \dots, d_n)$.

Plusieurs domaines peuvent être identiques. A chaque domaine est associé un nom d'*attribut* A_i qui est lié à sa signification sémantique par rapport à la relation.

Pour décrire une relation, on énumère ses attributs et on donne la liste des n -uplets qui lui appartiennent (définition en *extension*). On peut aussi définir le prédicat qui permet de juger si un n -uplet appartient à la relation (on dit parfois définition en *intension*).

Chaque n -uplet est une *occurrence* de la relation.

On note la relation $R(A_1, A_2, \dots, A_n)$

Dans la théorie relationnelle, tous les n -uplets doivent être distincts (sinon on ne pourrait les distinguer en tant qu'éléments d'un ensemble). Une relation a donc au moins un sous-ensemble $\{A_1, A_2, \dots, A_k\}$ de l'ensemble des attributs qui permet d'identifier chacun des n -uplets de la relation (l'ensemble des attributs satisfait cette condition) : deux éléments de la relation ne peuvent avoir les mêmes valeurs pour tous les attributs de ce sous-ensemble. S'il est minimal (au sens que, si on retire un des attributs, l'ensemble des attributs restants ne permettent plus d'identifier les n -uplets de la relation), un tel sous-ensemble d'attribut est appelé une *clé candidate*.

Le concepteur choisit une de ces clés candidates comme identifiant privilégié. La clé candidate choisie s'appelle alors la *clé primaire* de la relation.

On dira qu'un attribut est *non-clé* s'il n'appartient à aucune des clés candidates.

B. Tables

Une BD relationnelle est un ensemble de relations :

- relations de l'utilisateur (relations de table)
- relations de catalogue (relations système)

Propriétés d'une relation :

- nom unique dans l'ensemble des relations de la base
- nombre fixé de colonnes
- nombre de lignes quelconque

Exemple :

fournisseur	F#	Nom	Ville	pièce	P#	Nom	fournit	F#	P#
	F1	Smith	Londres		P1	vis		F1	P1
	F2	Meyer	Strasbourg		P5	écrou 6 pans		F1	P18
	F3	Muller	Strasbourg		P18	boulon		F3	P5
	F4	Ryan	Londres					F1	P18
	F5	Martin	Paris					F5	P18

1) Colonnes

Les colonnes sont appelées **champ** ou **attribut**

♦ Chaque attribut a un nom unique dans une relation donnée, mais on peut trouver deux attributs de même nom dans deux relations différentes.

Notation : fournisseur : F#
 fournit : F#

Pour le système, les deux F# sont totalement différents, mais sémantiquement, les deux sont identiques, on leur a donné le même nom.

- ♦ L'attribut prend sa valeur dans un **domaine** (ensemble des valeurs acceptées).

Exemple : ville $\in$ { Londres, Strasbourg, Paris, New York }

- ♦ L'ordre des attributs n'a pas d'importance.

Exemple : fournisseur < F#, Nom, Ville > = fournisseur < Nom, F#, Ville >

- ♦ Le nombre d'attributs définit le **degré** de la relation.

2) Lignes

Les lignes sont appelées **n-uplets** (tuples).

- les lignes sont supposées toutes différentes, il n'y a pas de doublon dans une table
- l'ordre des lignes n'a pas d'importance
- le nombre de n-uplets définit la **cardinalité**

C. Catalogue

Il existe un certain nombre de tables automatiquement gérées par le système (créées par celui-ci) :

- catalogue des relations (relations, créateur, statistiques sur le contenu, ...)
- catalogue des constituants (attributs, types, longueurs, ...)
- catalogue des chemins d'accès (index, attribut, type d'index, ...)
- catalogue des utilisateurs
- catalogue des droits d'accès
- etc.

Ces catalogues sont mis à jour par le SGBD.

Un utilisateur peut consulter les catalogues (même langage que pour les tables), mais n'a pas le droit d'écriture.

II. Langages d'interrogation

A. Les différents langages

Principalement trois théories qui ont donné lieu à différents langages :

- l'algèbre relationnelle $\rightarrow$ langage SQL
- calcul relationnel des tuples $\rightarrow$ langage QUEL (SGBD « Ingres »)
- calcul relationnel des domaines $\rightarrow$ langage QBE (*Query By Example*) d'IBM

On a pu démontrer qu'ils étaient équivalents dans le sens où ils permettent de désigner les mêmes ensembles de données.

B. Algèbre relationnelle

Algèbre relationnelle : collection d'opérations formelles qui agissent sur des relations et produisent des relations en résultat (E.F. Codd).

"L'algèbre relationnelle est aux relations ce que l'arithmétique est aux entiers."

Les opérations sont classées en deux catégories (Codd) : les opérations de base et les opérations dérivées.

Deux types d'opérations de base : opérations ensemblistes traditionnelles et les opérations spécifiques.

- opérations ensemblistes :

- union
- différence
- produit cartésien
- opérations spécifiques :
 - projection
 - restriction
 - jointure

Les opérations dérivées peuvent en général être obtenues par combinaison des opérations de base.

Opérations dérivées :

- intersection
- division
- complément
- éclatement
- jointure externe
- semi-jointure

1) Opérations ensemblistes

a) Union

Opération appliquée à deux relations de même schéma.

L'union de deux relations de même schéma R1 et R2 est la relation R3 ayant pour tuples les tuples de R1 ou ceux de R2.

Notation : $R1 \cup R2$

Exemple : fournisseur1 = < F#, Nom, Ville >, fournisseur2 = < F#, Nom, Ville >

fournisseur1	F#	Nom	Ville
	F1	Smith	Londres
	F2	Meyer	Strasbourg
	F3	Muller	Strasbourg

∪ fournisseur2

F#	Nom	Ville
F2	Meyer	Strasbourg
F4	Ryan	Londres

→ fournisseur

F#	Nom	Ville
F1	Smith	Londres
F2	Meyer	Strasbourg
F3	Muller	Strasbourg
F4	Ryan	Londres

b) Différence

Opération appliquée à deux relations de même schéma.

La différence entre deux relations de même schéma R1 et R2 est la relation R3 ayant pour tuples les tuples de R1 n'appartenant pas à R2.

Notation : $R1 - R2$

Exemple : fournisseur1 = < F#, Nom, Ville >, fournisseur2 = < F#, Nom, Ville >

fournisseur1	F#	Nom	Ville	– fournisseur2	F#	Nom	Ville
	F1	Smith	Londres		F2	Meyer	Strasbourg
	F2	Meyer	Strasbourg		F4	Ryan	Londres
	F3	Muller	Strasbourg				

→ fournisseur

F#	Nom	Ville
F1	Smith	Londres
F3	Muller	Strasbourg

Remarque : la différence est un opérateur non commutatif.

c) Produit cartésien (binaire)

Combinaison de deux relations (qui n'ont pas forcément le même schéma) pour en former une nouvelle qui a tous les attributs des deux relations d'origine.

En théorie, les deux relations ne doivent pas avoir d'attributs en commun.

En pratique, on renomme les attributs, par exemple en préfixant le nom de la relation (t.alias dans SQL).

Notation : $R1 \times R2$

Exemple : $R1(A,B), R2(C,D) \rightarrow R1 \times R2 = R(A,B,C,D)$

Exemple : fournisseur = < F#, Nom, Ville >, pièce = < P#, Nom >

fournisseur	F#	Nom	Ville	× pièce	P#	Nom
	F1	Smith	Londres		P1	vis
	F2	Meyer	Strasbourg		P5	écrou 6 pans
	F3	Muller	Strasbourg		P18	boulon

→ fournisseur × pièce

F#	fournisseur.Nom	Ville	P#	pièce.Nom
F1	Smith	Londres	P5	vis
F1	Smith	Londres	P1	écrou 6 pans
F1	Smith	Londres	P18	boulon
F2	Meyer	Strasbourg	P5	vis
F2	Meyer	Strasbourg	P1	écrou 6 pans
F2	Meyer	Strasbourg	P18	boulon
F3	Muller	Strasbourg	P5	vis
F3	Muller	Strasbourg	P1	écrou 6 pans
F3	Muller	Strasbourg	P18	boulon

Exemple : $R(A,B) \times R(A,B) \rightarrow$ problème de nom.

En SQL, on peut renommer les relations le temps de la requête.

2) Opérations spécifiques

a) Projection (unaire)

Opération spécifique aux relations qui permet de supprimer des attributs d'une relation, c'est-à-dire sélectionner une ou plusieurs colonnes d'une relation.

L'appellation provient du fait qu'on passe d'une relation n-aire à une relation p-aire, avec $p < n$ (projection d'un espace de dimension n dans un espace de dimension p).

Notation : $\text{proj}_Y(R)$, où R est une relation, l'ensemble de ses attributs est X, Y est inclus dans X.

Exemple : fournisseur X = {F#, Nom, Ville}

proj _{Ville} (fournisseur →	Ville	proj _{Nom, Ville} (fournisseur) →	Nom	Ville
	Londres Strasbourg Strasbourg		Smith Meyer Muller Ryan Martin	Londres Strasbourg Strasbourg Londres Paris

b) Restriction (unaire)

Egalement appelée sélection.

La relation résultat est de même type que la relation initiale.

Le résultat est obtenu en enlevant des tuples à la relation opérande selon un critère de sélection donné.

Notation : R|E, R est une relation, E est une expression logique.

E est du type <Attribut> θ <Valeur>.

θ ∈ {=, ≠, <, ≤, >, ≥}

Formule atomique : <Attribut> θ <Attribut>
 <Attribut> θ Constante>

Les formules atomiques peuvent être reliées par des connectivités ET, OU, NON.

Exemple : fournisseur = < F#, Nom, Ville >

fournisseur | ville = "Londres"

→ fournisseur	F#	Nom	Ville
	F1	Smith	Londres
	F4	Ryan	Londres

Exemple : stock < P#, Quantité-stock, Quantité-seuil >

stock | Quantité-stock < Quantité-seuil

c) Jointure (binaire)

Composition de deux relations à l'aide d'un critère de jointure.

♦ La jointure permet de calculer, à partir de deux relations R1(X, Y) et R2(Y, Z) une relation R(X,Y,Z) constituée des n-uplets qui sont combinaison de n-uplets de R1 et de R2 et qui ont même valeur sur l'attribut commun Y.

Notation : R1 * R2

Exemple : fournisseur = < F#, Nom, Ville >, fournit = < F#, P# >

fournisseur	F#	Nom	Ville	* fournit	F#	P#
	F1	Smith	Londres		F1	P1
	F2	Meyer	Strasbourg		F1	P18
	F3	Muller	Strasbourg		F3	P5
	F4	Ryan	Londres		F2	P18
	F5	Martin	Paris		F5	P18

→

F#	Nom	Ville	P#
F1	Smith	Londres	P1
F1	Smith	Londres	P18
F3	Meyer	Strasbourg	P18
F2	Muller	Strasbourg	P5
F5	Martin	Paris	P18

fournisseur * fournit = fournit $\times$ fournisseur | fournit.F# = fournisseur.F#

♦ Le θ -produit est défini en remplaçant l'égalité par n'importe quel opérateur de l'ensemble θ .

Remarque : dans certaines notations, le θ -produit est appelé inéqui-jointure (ou jointure naturelle) ; dans ce cas, la jointure est notée équi-jointure.

Remarque : lorsque plusieurs attributs sont communs aux deux relations, la comparaison porte sur toutes les relations.

3) Opérations dérivées

a) Intersection

Opération appliquée à deux relations de même schéma.

L'intersection de deux relations de même schéma R1 et R2 est la relation R3 ayant pour tuples ceux appartenant à la fois à R1 et à R2.

Notation : $R1 \cap R2$

Exemple : fournisseur1 = < F#, Nom, Ville >, fournisseur2 = < F#, Nom, Ville >

fournisseur1	F#	Nom	Ville	$\cap$ fournisseur2	F#	Nom	Ville
	F1	Smith	Londres		F2	Meyer	Strasbourg
	F2	Meyer	Strasbourg		F4	Ryan	Londres
	F3	Muller	Strasbourg				

→ fournisseur

F#	Nom	Ville
F2	Meyer	Strasbourg

Remarque : l'intersection est redondante parce qu'elle peut être obtenue par combinaison d'opérations de base :

$R1 \cap R2 = R1 - (R1 - R2)$, ou bien $R1 \cap R2 = R2 - (R2 - R1)$.

b) Division

La division permet de rechercher dans une relation les sous-tuples complétés par ceux d'une autre relation, répondant ainsi à des questions de la forme "quel que soit x, trouver y".

Notation : $R1 \div R2$

Exemple :

Nom	Ville	P#	÷	Ville	P#	→	Nom
Muller	Strasbourg	P1		Strasbourg	P18		Meyer
Muller	Strasbourg	P5		Strasbourg	P5		
Meyer	Strasbourg	P18					
Meyer	Strasbourg	P5					
Martin	Paris	P18					

Remarque : la division peut également être obtenue à partir de la composition d'autres opérations.

c) Complément

Le complément d'une relation est l'ensemble des tuples du produit cartésien des domaines d'une relation n'appartenant pas à cette relation.

Les domaines sont supposés finis (sinon on obtient des relations infinies).

Notation : $\neg R$

Exemple : fournit = $\langle F\#, P\# \rangle$

fournit	F#	P#	$\rightarrow \neg$ fournit	F#	P#
	F1	P1		F1	P5
	F1	P18		F2	P1
	F3	P5		F2	P5
	F2	P18		F3	P1
	F5	P18		F3	P18
d) Eclatement				F4	P1
				F4	P5
				F4	P18
				F5	P1
				F5	P5

d) Eclatement

L'éclatement fournit deux relations à partir d'une relation et d'une condition, la première vérifie la relation (= restriction), la seconde ne la vérifiant pas (= restriction avec l'opposé de la condition).

e) Jointure externe

Extension de la jointure sans perte d'information (en complétant par des champs nuls si nécessaire).

f) Semi-jointure

Jointure sans garder tous les champs des deux relations (= projection du résultat).

4) Propriétés de l'algèbre relationnelle

R0 : Commutativité et Associativité pour le produit cartésien

$$R \times S = S \times R$$

$$(R \times S) \times T = R \times (S \times T)$$

R1 : Eclatement d'une opération de Sélection – Commutativité des conditions de sélection

Considérons l'opération de sélection $R \mid F$ où F est une expression de sélection telle que :

$$F = F_1 \wedge F_2 \wedge F_3 \wedge \dots \wedge F_k$$

Eclatement :

$$\begin{aligned} R \mid F &= R \mid (F_1 \wedge F_2 \wedge F_3 \wedge \dots \wedge F_k) \\ &= (((R \mid F_1) \mid F_2) \dots \mid F_k) \end{aligned}$$

Commutativité :

Remarquons que $F = F_1 \wedge F_2 = F_2 \wedge F_1$. D'où

$$(R \mid F_1) \mid F_2 = (R \mid F_2) \mid F_1$$

R2 : Regroupement d'une série de projections en une seule opération de projection

Soit U l'ensemble des attributs de la relation R ,

Soient X et Y des sous-ensembles de U vérifiant de plus $X \subset Y$.

$$\text{proj}_X (\text{proj}_Y (R)) = \text{proj}_X (R)$$

R3 : Inversion sélection – projection

Si l'opération de sélection F définie sur les constituants X suit une opération de projection de R sur les constituants Y , on peut inverser les deux opérations si $X \subset Y$.

$$\text{proj}_Y(R) \mid F = \text{proj}_Y(R \mid F)$$

Dans le cas général :

$$\text{proj}_Y(R) \mid F = \text{proj}_Y(\text{proj}_{X \cup Y}(R) \mid F)$$

R4 : Inversion sélection – produit cartésien

Si F ne porte que sur des attributs de R :

$$(R \times S) \mid F = (R \mid F) \times S$$

Si $F = F_1 \wedge F_2$ où F_1 n'utilise que des attributs de R et F_2 que des attributs de S :

$$((R \times S) \mid F_1 \wedge F_2) = ((R \mid F_1) \times (S \mid F_2))$$

Si $F = F_1 \wedge F_2$ où F_1 n'utilise que des attributs de R mais F_2 utilise des attributs des deux relations :

$$((R \times S) \mid F) = ((R \mid F_1) \times S) \mid F_2$$

R5 : Inversion sélection – union

$$(R \cup S) \mid F = (R \mid F) \cup (S \mid F)$$

R6 : Inversion sélection – différence

$$\begin{aligned} (R - S) \mid F &= (R \mid F) - (S \mid F) \\ &= (R \mid F) - S \end{aligned}$$

La première solution est plus facile à calculer parce que la relation $(S \mid F)$ est plus petite que la relation S .

R7 : Inversion projection – produit cartésien

Soit X une liste d'attributs de R , Y une liste d'attributs de S et $Z = X \cup Y$. Alors,

$$\text{proj}_Z(R \times S) = \text{proj}_X(R) \times \text{proj}_Y(S)$$

Ne marche pas pour la composition.

R8 : Inversion projection – union

$$\text{proj}_X(R \cup S) = \text{proj}_X(R) \cup \text{proj}_X(S)$$

Remarque concernant R1 :

fournisseur | nom = Meyer ET ville = Strasbourg

$$= (\text{fournisseur} \mid \text{nom} = \text{Meyer}) \mid \text{ville} = \text{Strasbourg}$$

$$= (\text{fournisseur} \mid \text{ville} = \text{Strasbourg}) \mid \text{nom} = \text{Meyer}, \text{ par commutativité}$$

Remarque concernant R2 :

$$\text{proj}_{\text{Nom}}(\text{proj}_{\text{Nom}, \text{Ville}}(\text{fournisseur}))$$

$$= \text{proj}_{\text{Nom}}(\text{fournisseur}), \text{ parce que } \{\text{Nom}\} \in \{\text{Nom}, \text{Ville}\}$$

Par contre,

$$\text{proj}_{P\#}(\text{proj}_{\text{Nom}, \text{Ville}}(\text{fournisseur}))$$

$$\neq \text{proj}_{P\#}(\text{fournisseur})$$

5) Le langage SQL

Structured Query Language.

Requête :

SELECT ... (attributs sur lesquels on projette)
FROM ... (relation, ou produit cartésien)
[WHERE ...] (sélection, optionnel)

a) Exemple fournisseur / pièce / fournit

fournisseur(F#, Nom, ville)
pièce(P#, Nom)
fournit(F#, P#)

proj_{Ville}(fournisseur)

→ SELECT Ville
FROM fournisseur

proj_{Nom, Ville}(fournisseur)

→ SELECT Nom, Ville
FROM fournisseur

fournisseur | Ville = Londres

→ SELECT F#, Nom, Ville (= SELECT *)
FROM fournisseur
WHERE ville = 'Londres'

proj_{P#}(fournisseur × fournit | fournisseur.F# = fournit.F# et ville = Londres)

→ SELECT P#
FROM fournisseur, fournit
WHERE fournisseur.F# = fournit.F#
AND ville = 'Londres'

b) Exemple joueur / rencontre / gain

joueur(nom, prénom, âge, pays)
rencontre(nrencontre, gagnant, perdant, lieu-tournoi, date, score)
gains(nom-joueur, lieu-tournoi, date, prime, sponsor)

1. Nom et âge des joueurs américains, autre que 'Connors', âgés de plus de 30 ans.

proj_{âge, nom}(joueur | pays = 'USA' ET nom ≠ 'Connors' ET âge > 30)

→ SELECT nom, âge FROM joueur
WHERE pays = 'USA' AND nom != 'Connors' AND âge > 30

2. Les pays des joueurs de 20 ans (sauf France)

proj_{pays}(joueur | pays ≠ 'France' ET âge = 20)

→ SELECT DISTINCT pays FROM joueur
WHERE pays != 'France' AND âge = 20

3. Nom et âge des joueurs ayant participé à Roland Garros en 99.

proj_{nom, âge} joueur *_{nom = nom-joueur} gains | lieu-tournoi = 'RG' ET date = 99

Première solution en rapport direct avec l'expression algébrique :

→ SELECT nom, âge FROM joueur, gains
WHERE nom = nom-joueur AND lieu = 'RG' AND date = 99

Seconde solution, plus intuitive, plus en rapport avec la question exprimée en langage courant :

→ SELECT nom, âge FROM joueur
WHERE nom IN (SELECT nom-joueur FROM gains
WHERE lieu = 'RG' AND date = 99)

4. Nom et pays des joueurs ayant participé, en 95, aux tournois de Rolland Garos et Wimbledon.

$\text{proj}_{\text{nom,pays}} \text{joueur} *_{\text{nom} = \text{nom-joueur}} \text{gains} \mid \text{lieu-tournoi} = \text{'RG'} \text{ ET } \text{date} = 95$

$\cap$

$\text{proj}_{\text{nom,pays}} \text{joueur} *_{\text{nom} = \text{nom-joueur}} \text{gains} \mid \text{lieu-tournoi} = \text{'Wimbledon'} \text{ ET } \text{date} = 95$

Première solution, déduite intuitivement de la question :

```
→ SELECT nom, pays FROM joueur
    WHERE nom IN (SELECT nom-joueur FROM gains
                  WHERE lieu = 'RG' AND date = 95)
    AND nom IN (SELECT nom-joueur FROM gains
                WHERE lieu = 'Wimbledon' AND date = 95)
```

Seconde solution, en rapport avec l'expression algébrique, et utilisant un mot-clé de SQL :

```
→ SELECT nom, pays FROM joueur, gains
    WHERE nom = nom-joueur AND lieu = 'RG' AND date = 95
INTERSECT
SELECT nom, pays FROM joueur, gains
    WHERE nom = nom-joueur AND lieu = 'Wimbledon' AND date = 95
```

5. Nom des joueurs ayant perdu toutes leurs rencontres.

On utilise pour cela la relation *rencontre*, dans laquelle les attributs *gagnant* et *perdant* représentent des noms de joueurs.

La seule sélection de la colonne *perdant* est insuffisante pour déterminer les louseurs, parce qu'un perdant peut apparaître gagnant d'un autre match.

Il faut donc soustraire les gagnants des perdants :

$\text{proj}_{\text{perdant}} \text{rencontre} - \text{proj}_{\text{gagnant}} \text{rencontre}$

Première solution utilisant un mot-clé de SQL :

```
→ SELECT DISTINCT perdant FROM rencontre
MINUS
SELECT DISTINCT gagnant FROM rencontre
```

Seconde solution, plus intuitive :

```
→ SELECT DISTINCT perdant FROM rencontre
    WHERE perdant NOT IN (SELECT DISTINCT gagnant FROM rencontre)
```

C. Calcul relationnel

Le calcul relationnel des tuples et le calcul relationnel des domaines reposent sur le calcul des prédicats du premier ordre. Cette théorie mathématique étudie les formules logiques construites avec un ensemble de prédicats (c'est-à-dire de propositions qui peuvent être vraies ou fausses dans un certain contexte), les opérateurs *et*, *ou*, *négation*, *implication logique*, des variables et les opérateurs $\forall$ et $\exists$.

A chaque formule logique correspond l'ensemble des données qui vérifient cette formule. L'interrogation de la base de données consiste donc à énoncer une formule qui correspond aux données que l'on souhaite extraire de la base.

La différence essentielle entre le calcul relationnel des tuples et le calcul relationnel des domaines est l'ensemble dans lequel les variables prennent leurs valeurs :

- tuples : chaque variable prend ses valeurs dans l'ensemble des tuples d'une relation particulière
- domaines : chaque variable prend ses valeurs dans un domaine particulier des attributs de la base.

Opérateurs utilisés :

- prédicats : opérateurs habituels ($<$, $>$, $\leq$, $\geq$, $\neq$)
- tuples : on ajoute l'opérateur unaire qui indique qu'une variable prend ses valeurs dans une relation (par exemple fournisseur(V) indique que la variable V appartient à la relation fournisseur)
- domaines : on ajoute l'opérateur unaire qui indique qu'une variable prend ses valeurs dans un domaine (par exemple fournisseur(F#.V) indique que la variable V prend ses valeurs dans le domaine de l'attribut F# de la relation fournisseur)

Exemple :

Pièces fournies par Smith :

tuples : {P.Nom / pièce(P) et $\exists F$ fournisseur(F) et (F.Nom = 'Smith') et...

domaines : ...

Nom des fournisseurs (qui fournissent effectivement des pièces) :

tuples : {F.Nom / fournisseur(F) et ($\exists R$ fournit(R) et F.F# = R.F#)}

domaines : {n / $\exists m$ fournisseur(F# : m, Nom : n) et fournit(F# : m)}

III. Dépendances fonctionnelles

But : Construire le schéma conceptuel.

Tout est construit à partir des Dépendances fonctionnelles (DF), qui sont des contraintes d'intégrité très fortes.

Le traitement sera différent suivant qu'il y a une ou plusieurs réponses à la question posée.

A. Définition

Soit R une relation dont les attributs sont $U = \{u_1, u_2, \dots, u_n\}$ et $X \subset U, Y \subset U$.

On dit que dans la relation, Y dépend fonctionnellement de X, ou X détermine Y, si :

$\forall u \in R, \forall v \in R, \text{proj}_X(R | u) = \text{proj}_X(R | v) \Rightarrow \text{proj}_Y(R | u) = \text{proj}_Y(R | v)$.

Plus simplement, un attribut Y dépend fonctionnellement d'un attribut X si, étant donnée une valeur de X, il lui correspond une valeur unique de Y (quel que soit le n-uplet considéré).

Notation : $X \rightarrow Y$

u_1	u_2	u_3	...	...	u_n
a	b				
a	b				

Posons par exemple $X = u_1$ et $Y = u_2$.

Si deux lignes ont la même valeur sur X, alors ils ont la même valeur sur Y : dans ce cas, $X \rightarrow Y$.

Exemple :

n-ch	type	nb-lits	douche
1	A	2	oui
2	B	1	oui
3	A	2	non
4	A	2	non
5	B	1	oui
6	C	3	non

Dans ce cas, les DF sont :

type $\rightarrow$ nb-lits

n-ch $\rightarrow$ type

n-ch $\rightarrow$ nb-lit

n-ch $\rightarrow$ douche

nb-lits $\rightarrow$ type

Exemple :

< Nom, Prénom, Ville, CP, Département >

DF : CP $\rightarrow$ Département

(mais pas Ville $\rightarrow$ Département parce que 2 villes peuvent avoir le même nom)

Exemple :

< Nom, Prénom, Moyenne, Âge, Enseignant >

DF : Nom, Prénom $\rightarrow$ Moyenne

Nom, Prénom $\rightarrow$ Âge

A condition de se placer dans une classe d'étudiants sans homonymes (pour le nom et le prénom).

B. Règles d'inférence

Notation : $\Rightarrow$ (en principe $\perp$ tourné vers la droite...)

Il est généralement possible de déduire de nouvelles DF à partir des DF existantes.

Notons X, Y, Z et W des sous-ensembles d'attributs de la relation R.

Les trois premières règles sont les axiomes d'Armstrong (1974), les trois suivantes en sont déduites.

1) Axiomes d'Armstrong*a) Réflexivité*

Tout ensemble d'attributs détermine lui-même ou une partie de lui-même.

$Y \subseteq X \Rightarrow X \rightarrow Y$

Exemple :

$X = \{\text{Nom, Catégorie}\}$

$Y = \text{Nom}$ ou $Y = \{\text{Catégorie}\}$ ou $Y = \{\text{Nom, Catégorie}\}$

Nom, Catégorie $\rightarrow$ Nom

Nom, Catégorie $\rightarrow$ Catégorie

b) Augmentation

Si X détermine Y, les deux ensembles d'attributs peuvent être enrichis par un troisième.

$X \rightarrow Y \Rightarrow X, Z \rightarrow Y, Z$

$Z \subseteq W$ et $X \rightarrow Y \Rightarrow X, W \rightarrow Y, Z$

c) Transitivité

Le composé de deux fonctions dont l'image de l'une est le domaine de l'autre, est une fonction.

$X \rightarrow Y$ et $Y \rightarrow Z \Rightarrow X \rightarrow Z$

2) Règles complémentaires*a) Union*

$X \rightarrow Y$ et $X \rightarrow Z \Rightarrow X \rightarrow Y, Z$

b) Pseudo-transitivité

$X \rightarrow Y$ et $Y, W \rightarrow Z \Rightarrow X, W \rightarrow Z$

c) Décomposition

$X \rightarrow Y$ et $Z \rightarrow Y \Rightarrow X \rightarrow Z$

(parce que $Z \subseteq Y \Rightarrow Y \rightarrow Z$)

C. Manipulation des DF

1) Ajout / suppression d'attributs

- supposons $G1, G2 \rightarrow D1, D2$ VRAIE.
- ajout d'attributs à gauche : $X, G1, G2 \rightarrow D1, D2$
- ajout d'attributs à droite : NON (à moins qu'une autre DF le permette)
- suppression d'attributs à gauche : NON (à moins qu'une autre DF le permette)
- suppression d'attributs à droite : $G1, G2 \rightarrow D1$

Choix des DF à retenir :

Il faut parfois éviter de trop simplifier par transitivité, pour éviter de perdre des DF. $X \rightarrow Y$ et $Y \rightarrow Z \Rightarrow X \rightarrow Z$. Les deux premières DF permettent de déduire la troisième, le contraire est faux. Il vaut donc mieux garder les deux premières.

$G1, G2 \rightarrow X$ et $G1 \rightarrow X$. La seconde permet de déduire la première, le contraire est faux. On garde donc la seconde.

2) Fermeture

Soit F un ensemble de DF. Il est possible de composer par les règles d'inférence les DF de F . La fermeture transitive de F , notée F^+ , est l'ensemble de toutes les DF résultant de l'application des règles d'inférence sur les DF de F .

L'application d'une règle d'inférence construit F_1 à partir de F , puis F_2 , ..., puis F_n . On s'arrête lorsque $F_{n+1} = F_n$, et donc $F^+ = F_n (= F_{n+1})$.

Deux ensembles de DF F_1 et F_2 sont équivalents s'ils ont la même fermeture, c'est-à-dire $F_1^+ = F_2^+$

$$\begin{array}{ll} F_1 \Rightarrow F_2 & F_1^+ \supseteq F_2^+ \\ F_2 \Rightarrow F_1 & F_2^+ \supseteq F_1^+ \end{array}$$

3) Couverture

Il est intéressant de déterminer un sous-ensemble minimal de DF permettant de générer toutes les autres : c'est la couverture minimale d'un ensemble de DF.

La couverture minimale de F est l'ensemble F_0 tel que : $F_0^+ = F^+$.

Notation : $\wedge F$

Exemple :

$$\begin{array}{ll} F = \{ & Q \rightarrow S \\ & T \rightarrow R \\ & R, S \rightarrow P \\ & T, Q \rightarrow P \\ & P, R \rightarrow S \} & F' = \{ & T \rightarrow R \\ & R, S \rightarrow P \\ & T, Q \rightarrow P \\ & P, R \rightarrow S \} \end{array}$$

Est-ce qu'on arrive à retrouver $Q \rightarrow S$ avec F' ? NON, donc $Q \rightarrow S$ nécessaire.

Même démarche pour chaque DF.

On peut supprimer $T, Q \rightarrow P$:

$Q \rightarrow S \Rightarrow Q, T \Rightarrow S$ (augmentation)

$T \rightarrow R \Rightarrow T, Q \rightarrow R$ (augmentation)

$Q, T \rightarrow S$ et $T, Q \rightarrow R \Rightarrow T, Q \rightarrow R, S$ (union)

$T, Q \rightarrow R, S$ et $R, S \rightarrow P \Rightarrow T, Q \rightarrow P$ (transitivité)

donc $\wedge F = \{$

- $Q \rightarrow S$
- $T \rightarrow R$
- $R, S \rightarrow P$
- $P, R \rightarrow S\}$

Le calcul de la couverture permet de faire baisser le nombre de vérifications qui sont à faire.

4) DF élémentaires

Une DF $X \rightarrow Y$ est **pleine** ou **élémentaire** s'il n'existe pas de sous-ensemble strict X' de X ($X' \subset X$ et $X' \neq X$) tel que $X' \rightarrow Y$.

On dira que Y est pleinement dépendant de X .

5) Clé de relation

Il est possible de donner une définition formelle de la notion de clé à partir des DF : une clé est un ensemble minimal d'attributs qui détermine tous les autres.

Autrement dit, X est un clé (ou un identifiant) de la relation R si $X \rightarrow U$ est une DF élémentaire.

Un ensemble d'attributs qui inclut une clé est appelé **superclé**.

Il peut y avoir plusieurs clés pour une relation, mais on en choisit une comme clé primaire.

Notation : $\langle \underline{\text{Numéro}}, \text{Nom} \rangle$ (la clé est Numéro).

Lorsque deux attributs forment la DF élémentaire : $\langle \underline{\text{Num1}}, \underline{\text{Num2}}, \text{Nom} \rangle$ (la clé est le couple). La notation $\langle \underline{\text{Num1}}, \underline{\text{Num2}}, \text{Nom} \rangle$ est identique.

D. Passage DF - schéma relationnel : cas pratique

Une base de données sur les gîtes d'une région donnée comporte les attributs et les hypothèses suivants :

nom de l'attribut	type	descriptif
nom	chaîne	nom de l'hôtel (unique dans une région donnée)
catégorie	entier	nombre d'étoiles de l'hôtel
nb-chambres	entier	nombre total de chambres pour un hôtel donné
num-chambre	entier	numéro de chambre
type	entier	type de chambre
prix	réel	prix d'une chambre
nbre-lits	entier	nombre de lits
douche	booléen	vrai si la chambre comporte une douche privée
baaignoire	booléen	vrai si la chambre comporte une baignoire privée
WC	booléen	vrai si la chambre comporte des WC privés
lib-div	chaîne	libellé des divertissements proposés par l'hôtel
gratuit	booléen	indique si un divertissement proposé par un hôtel est gratuit ou payant
saison	{'H','B'}	H : haute saison, B : basse saison

Le type de chambre détermine les commodités offertes : nombre de lits, présence de douche, de baignoire, de WC indépendants. Le prix d'une chambre dépend du type de la chambre. Ainsi, dans un hôtel donné, toutes les

chambres de même type auront le même prix. Cette hypothèse n'est pas vraie pour deux hôtels différents. Le prix d'une chambre dépend également de la saison : haute ou basse.

Les dépendances fonctionnelles vous semblent-elles correctes ?

1. nom $\rightarrow$ catégorie, nb-chambres
2. num-chambre $\rightarrow$ type
3. type $\rightarrow$ prix
4. type $\rightarrow$ nbre-lits
5. type, prix $\rightarrow$ douche, baignoire, WC
6. nom, catégorie $\rightarrow$ nb-chambres
7. nom, num-chambre $\rightarrow$ type, douche, baignoire, WC
8. nom, gratuit $\rightarrow$ lib-div
9. catégorie, lib-div $\rightarrow$ gratuit
10. nom, saison, type $\rightarrow$ nbre-lits

1. Réponses :

- | | |
|--------|---------|
| 1. OUI | 6. OUI |
| 2. NON | 7. OUI |
| 3. NON | 8. NON |
| 4. OUI | 9. NON |
| 5. OUI | 10. OUI |

2. Chercher toutes les DF :

D'abord les plus simples :

- nom $\rightarrow$ catégorie
- nom $\rightarrow$ nb-chambre
- nom $\rightarrow$ lib-div
- type $\rightarrow$ nb-lit
- type $\rightarrow$ douche
- type $\rightarrow$ baignoire
- type $\rightarrow$ WC

Puis les autres :

- nom $\rightarrow$ catégorie, nb-chambre
- type $\rightarrow$ douche, baignoire, WC, nb-lit
- nom, num-chambre $\rightarrow$ type, douche, baignoire, WC
- nom, type, saison $\rightarrow$ prix
- nom, lib-div $\rightarrow$ gratuit

3. Déterminer F_0 :

- $F_0 = \{$
- nom $\rightarrow$ catégorie, nbre-chambre
 - nom, num-chambre $\rightarrow$ type
 - type $\rightarrow$ nb-lit, douche, baignoire, WC
 - nom, lib-div $\rightarrow$ gratuit
 - nom, type, saison $\rightarrow$ prix
- $\}$

4. Transformation en modèle relationnel :

Généralement, on a une relation par DF.

On obtient :

HOTEL < nom, catégorie, nbre-chambre >
CHAMBRE < nom, num-chambre, type >
COMMODITE < type, nb-lit, douche, baignoire, WC >
DIVERTISSEMENT < nom, lib-div, gratuit >
COUT < nom, type, saison, prix >

IV. Normalisation de relation

A. Principe

Problème :

- minimiser les redondances
- minimiser les anomalies de mise à jour

On part de deux ensembles :

- une relation universelle : relation unique qui contient tous les attributs de l'application,
- l'ensemble F des DF sous forme minimale.

On décompose la relation universelle en utilisant F pour obtenir un ensemble de relations pour une certaine forme normale (FN) choisie à l'avance : 1^{ère}, 2^{ème}, 3^{ème}, 3.5^{ème} (forme normale de Boyce Codd), 4^{ème} et 5^{ème}.

Les trois premières FN ont été définies par Codd (1972).

- 1FN : la plus simple, évite les domaines composés de plusieurs valeurs.
- 2FN : beaucoup de redondances → coût élevé de MAJ.
- 3FN : peu de redondances → coût faible de MAJ, mais beaucoup de jointures.

Exemple :

base étudiant

net : numéro d'étudiant

nom

âge

adresse

tarif : tarif CROUS $\in \{T_1, T_2, T_3\}$ selon l'âge de l'étudiant

matière

enseignant : enseignant responsable de la matière

moyenne : moyenne dans une matière donnée

ETUDIANT < net, nom, âge, adresse, tarif, matière, enseignant, moyenne >
--

DF : net → nom, âge, adresse
matière → enseignant
net, matière → moyenne
âge → tarif

Clé : net, matière (pas âge car net → âge → tarif).

Vérifier qu'aucun attribut ne manque.

Redondance et anomalie de mise à jour :

net	nom	âge	adresse	tarif	matière	enseignant	moyenne
12	Meyer	25	Strasbourg	T ₁	UV1	Brandel	NULL
12	Meyer	25	Strasbourg	T ₁	UV2	Tellier	10
40	Muller	18	Brumath	T ₂	UV3	Agnus	14

- Insertion, suppression : toujours une ligne complète.
- Redondance si Meyer prend plusieurs UV.
- Si Meyer change d'adresse, il faut la changer dans toutes les lignes.
- Effet de bord : si Muller abandonne l'UV3 et qu'on supprime son inscription, alors on perd toutes les informations qui le concerne s'il n'est pas inscrit dans une autre UV.

Exemple :

Relation employés < matricule, nom, dpt, nom-dpt > avec la DF dpt → nom-dpt

Anomalies lors de modifications des données :

- Anomalies d'**insertion** : on ne peut pas ajouter de nouveaux départements (numéro et nom) si le département ne comporte pas d'employé.
- Anomalies de **modification** : si on change le nom d'un département, on doit le changer dans toutes les lignes des employés de ce département.
- Anomalies de **suppression** : si on supprime le dernier employé d'un département, on perd l'information (numéro et nom) du département.

→ on découpe les tables en formes normales.

B. Décomposition en formes normales

1) 1^{ère} forme normale (1FN)

Justifiée par la simplicité et l'esthétique, la 1FN consiste à éviter les domaines constitués de plusieurs valeurs.

On dit qu'une relation est en 1FN si tous ses attributs sont de type élémentaires : char, string, int, float.

Interdit : les attributs de type structure, ensemble et liste.

Si on a besoin d'une structure → mise à plat de la structure, ou nouvelle relation.

Exemple :

net	adresse				nom
	numéro	rue	ville	CP	

→ INTERDIT

Soit mise à plat :

net	numéro	rue	ville	CP	nom

Soit nouvelle relation :

net	adresse	nom
	1 2	

et

id-adr	rue	ville	CP
1 2			

Si on a un ensemble (tous les éléments distincts), on duplique :

net	matière	note
1	UV1	{12, 11, 16}

INTERDIT →

net	matière	note
1	UV1	12
1	UV1	11
1	UV1	16

(clé = net, mat, note)

Si on a une liste (différent d'un ensemble car l'ordre a une importance et on peut avoir plusieurs fois la même valeur), il faut rajouter un attribut :

net	matière	note
1	UV1	{12, 11, 12}

INTERDIT →

net	matière	note	ordre
1	UV1	12	1
1	UV1	11	2
1	UV1	12	3

Remarque : chaque valeur du domaine est atomique, mais du point de vue du modèle relationnel : une date ou une image peut être considérée comme atomique, tout dépend du niveau de décomposition.

Exemple : base étudiant

ETUDIANT <net, nom, âge, adresse, tarif, matière, enseignant, moyenne>
--

En 1FN ? OUI.

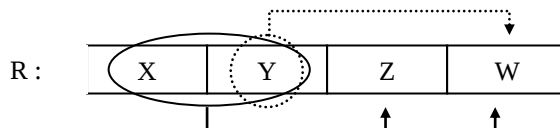
2) 2^{ème} forme normale (2FN)

La 2FN permet d'éliminer certaines redondances en assurant qu'aucun attribut n'est déterminé par une partie seulement de la clé.

Une relation est en 2FN si et seulement si :

- elle est en 1FN
- les attributs qui ne sont pas clé sont pleinement dépendant de chaque clé.

Exemple typique de relation NON en 2FN :



La clé est le couple X, Y. Donc $X, Y \rightarrow Z, W$. Or Y détermine à lui seul W : $Y \rightarrow W$. Donc W dépend d'une partie de la clé. Il faut donc décomposer R en : R1 <X, Y, Z> et R2 <Y, W>.

Exemple : base étudiant

ETUDIANT <net, nom, âge, adresse, tarif, matière, enseignant, moyenne>
--

ETUDIANT est en 1FN.

La clé est le couple net, matière.

net, matière → nom	pleine ? NON, car nom dépend uniquement de net
net, matière → âge	pleine ? NON, car âge dépend uniquement de net
net, matière → adresse	pleine ? NON, car adresse dépend uniquement de net
net, matière → tarif	pleine ? NON, car tarif dépend uniquement de net
net, matière → enseignant	pleine ? NON, car enseignant dépend uniquement de matière
net, matière → moyenne	pleine ? OUI, car moyenne dépend des deux

→ pas en 2FN.

→ on découpe :

net → nom, âge, adresse, tarif
 matière → enseignant
 net, matière → moyenne

Et on obtient le schéma relationnel :

ETUDIANT	< <u>net</u> , nom, âge, adresse, tarif>
ENSEIGNEMENT	< <u>matière</u> , enseignant>
NOTE	< <u>net, matière</u> , moyenne>

En 2FN ? OUI.

Remarques :

Si on supprime une moyenne, on ne perd pas d'étudiant.

Par contre, il faut fixer une Contrainte d'Intégrité référentielle (CI réf) exprimant que si on supprime un étudiant, il faut que la moyenne disparaisse aussi :

CI réf : $\text{proj}_{\text{net}}(\text{NOTE}) \subset \text{proj}_{\text{net}}(\text{ETUDIANT})$
 $\text{proj}_{\text{matière}}(\text{NOTE}) \subset \text{proj}_{\text{matière}}(\text{ENSEIGNEMENT})$

Exemple de mauvais découpage :

Relation universelle : $R = \langle \underline{A}, B, C \rangle$, découpée en $R_1 = \langle A, B \rangle$ et $R_2 = \langle A, C \rangle$.

A	B	C
a ₁	b ₁	c ₁
a ₁	b ₂	c ₂

→

A	B
a ₁	b ₁
a ₁	b ₂

et

A	C
a ₁	c ₁
a ₁	c ₂

Si on effectue le produit cartésien : $R_1 \times R_2$:

A	B	C
a ₁	b ₁	c ₁
a ₁	b ₁	c ₂
a ₁	b ₂	c ₁
a ₁	b ₂	c ₂

→ FAUX.

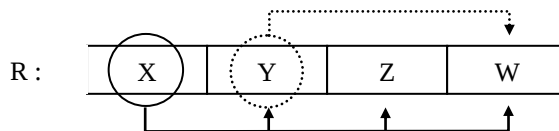
3) 3^{ème} forme normale (3FN)

La 3FN permet d'assurer l'élimination des redondances dues aux dépendances transitives.

Une relation est en 3FN si et seulement si :

- Elle est en 2FN.
- Tout attribut n'appartenant pas à une clé ne dépend pas d'un autre attribut non clé. Autrement dit, tous les attributs n'appartenant à aucune clé sont directement et non transitivement dépendants d'une clé.

Exemple typique de relation NON en 3FN :



La clé de la relation est X. Donc $X \rightarrow Y, Z, W$. Or Y détermine à lui seul W : $Y \rightarrow W$. Donc W dépend d'un attribut non clé. Il faut donc décomposer R en : $R_1 \langle \underline{X}, Y, Z \rangle$ et $R_2 \langle \underline{Y}, W \rangle$.

Exemple : base étudiant

La relation ETUDIANT $\langle \underline{\text{net}}, \text{nom}, \text{âge}, \text{adresse}, \text{tarif} \rangle$ n'est pas en 3FN parce que $\text{net} \rightarrow \text{âge} \rightarrow \text{tarif}$.

Donc on découpe cette relation en ETUDIANT $\langle \underline{\text{net}}, \text{nom}, \text{âge}, \text{adresse} \rangle$ et PRIX $\langle \underline{\text{âge}}, \text{tarif} \rangle$.

Les autres relations sont déjà en 3FN.

Le schéma relationnel étudiant devient donc :

ETUDIANT $\langle \underline{\text{net}}, \text{nom}, \text{âge}, \text{adresse} \rangle$
 PRIX $\langle \underline{\text{âge}}, \text{tarif} \rangle$
 ENSEIGNEMENT $\langle \underline{\text{matière}}, \text{enseignant} \rangle$
 NOTE $\langle \underline{\text{net}}, \underline{\text{matière}}, \text{moyenne} \rangle$

En 3FN ? OUI.

CI réf à ajouter pour éviter des redondances en cas de suppression dans la relation ETUDIANT :

$\text{proj}_{\text{âge}}(\text{PRIX}) \subset \text{proj}_{\text{âge}}(\text{ETUDIANT})$

4) Forme normale de Boyce-Codd

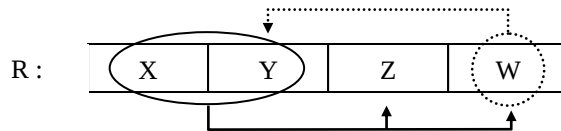
La 2FN élimine les anomalies créées par des DF entre parties de clé et attribut non clé.

La 3FN élimine les anomalies créées par des DF entre les attributs non clés.
 Et les DF de parties de clés entre elles ou d'attributs non clé vers une partie de clé ?
 → 3FN insuffisante → FNBC.

Une relation est en FNBC si et seulement si les seules DF élémentaires sont celles dans lesquelles une clé entière détermine un attribut.

En clair, pas de DF autres que $K \rightarrow A$, K étant la clé et A un attribut non clé.

Exemple typique de relation NON en FNBC :



La clé est le couple X, Y. Donc $X \rightarrow Z, W$. Or W détermine X : $W \rightarrow X$. Donc une partie de la clé dépend d'un attribut non clé. Il faut donc décomposer R en : R1 <X, Y, Z> et R2 <W, X>.

Exemple : base étudiant

Le schéma décomposé en 3FN est-il en FNBC ? OUI !

Exemple : base vins

LOCALISATION <cru, pays, région, qualité>

clé : cru, pays

DF : { cru, pays $\rightarrow$ région
 cru, pays $\rightarrow$ qualité
 région $\rightarrow$ pays }

cru	pays	région	qualité
Juliéna	France	Beaujolais	TB
Morgon	France	Beaujolais	TB
Chablis	US	Californie	M
Brouilly	US	Californie	M
Chablis	France	Beaujolais	M

En 3FN, parce que le cru ne détermine pas la région (il paraît !), mais pas en FNBC.

On décompose :

CRUS <cru, pays, qualité>
 REGION <région, pays>

DF : { cru, pays $\rightarrow$ qualité
 région $\rightarrow$ pays }

La décomposition est sans perte, malgré la perte d'une DF.

5) 4^{ème} forme normale (4FN)

Utilisation des dépendances multivaluées (DM, notées $X \twoheadrightarrow Y$), pour supprimer certaines redondances.

Exemple : base étudiant

Soit la relation :

net	matière	sport
1	BD	escalade
1	BD	tennis
1	Algo	escalade
1	Algo	tennis
2	Algo	VTT

clé : net, cours, sport

Il est possible de suivre plusieurs matières et de pratiquer plusieurs sports.

Les redondances apparaissent clairement, mais il n'est pas possible de décomposer parce qu'il n'y a pas de DF le permettant.

On utilise pour cela les dépendances multivaluées : indépendance entre deux ensembles d'attributs corrélés par un troisième.

La 4FN est obtenue en décomposant la relation à partir des DM.

DM : { net $\twoheadrightarrow$ matière
net $\twoheadrightarrow$ sport }

La relation est décomposée en :

COURS <net, cours> SPORT <net, sport>
--

6) 5^{ème} forme normale (5FN)

Utilisation des dépendances de jointure (DJ), pour supprimer certaines redondances.

Exemple : base vins

buveur	cru	producteur
Meyer	Edel	Claude
Meyer	Edel	André
Meyer	Gewurtz	André
Brandel	Edel	André

En 4FN parce qu'il n'existe pas de DM (par exemple buveur $\twoheadrightarrow$ cru est faux parce qu'il n'existe pas le tuple (Meyer, Gewurtz, Claude)).

Mais il reste encore des redondances : on apprend deux fois que Meyer boit de l'Edel et que André produit de l'Edel.

Mais il est impossible de décomposer de deux relations.

Les DM permettent de décomposer une relation en deux, les DJ permettent de décomposer une relation en N relations.

On obtient alors le schéma :

buveur	cru
Meyer	Edel
Meyer	Gewurtz
Brandel	Edel

buveur	producteur
Meyer	Claude
Meyer	André
Brandel	André

cru	producteur
Edel	Claude
Edel	André
Gewurtz	André

En 5FN.

C. Exemples

R <A, B, C>

Donnez le schéma conceptuel de la BD en 3FN dans les trois cas suivants :

1. $F = \{A \rightarrow B, B \rightarrow C\}$
2. $F = \{A \rightarrow B, C\}$
3. $F = \{\}$

1. 1FN ? OUI

R <A, B, C>, clé : A

2FN ? OUI

3FN ? NON, car transitif

R1 <A, B>, R2 <B, C> en 3FN.

2. 1FN ? OUI

R <A, B, C>, clé : A

2FN ? OUI

3FN ? OUI (découpage possible R1 <A, B>, R2 <A, C> mais inutile).

3. 1FN ? OUI

R <A, B, C>, clé : A, B, C

2FN ? OUI (ne surtout pas décomposer)

3FN ? OUI

Exemple des gîtes

GITES < nom, nb-chambres, num-chambre, type, prix, nbre-lits, douche, baignoire, WC, lib-div, gratuit, saison >

$F_0 = \{$
 nom $\rightarrow$ catégorie, nbre-chambres
 nom, num-chambre $\rightarrow$ type
 type $\rightarrow$ nb-lits, douche, baignoire, WC
 nom, lib-div $\rightarrow$ gratuit
 nom, type, saison $\rightarrow$ prix $\}$

En 1FN ? OUI (tous les attributs sont atomiques).

En 2FN ? NON, parce que par exemple, nb-chambres ne dépend que de nom, et donc n'est pas pleinement dépendant de la clé.

→ on décompose :

R1 <nom, catégorie, nbre-chambres>
R2 <nom, num-chambres, type, nb-lits, douche, baignoire, WC>
R3 <nom, lib-div, gratuit>
R4 <nom, type, saison, prix>

En 2FN ? OUI, tous les attributs sont pleinement dépendants de la clé.

En 3FN ? NON, parce que dans R2, type définit nb-lits par exemple, et donc un attribut non clé définit un autre attribut non clé.

→ on décompose :

R1 < <u>nom</u> , catégorie, nbre-chambres>
R21 < <u>nom</u> , <u>num-chambres</u> , type>
R22 < <u>type</u> , nb-lits, douche, baignoire, WC>
R3 < <u>nom</u> , <u>lib-div</u> , gratuit>
R4 < <u>nom</u> , <u>type</u> , <u>saison</u> , prix>

En 3FN ? OUI.

En FNBC ? OUI.

V. Insertion - suppression

Emploi de n-uplets, sans précision sur son mode d'obtention (entrée au clavier + vérification).

Couples pré-conditions - post-conditions :

- pré-condition : doit être vérifiée pour avoir le droit d'exécuter l'opération
- post-condition : le résultat de l'opération doit la vérifier

Exemple : base étudiants

Attributs : net, nom, âge, adresse, tarif, matière, enseignant, moyenne

ETUDIANT < <u>net</u> , nom, âge, adresse>
PRIX < <u>âge</u> , tarif>
ENSEIGNEMENT < <u>matière</u> , enseignant>
NOTE < <u>net</u> , <u>matière</u> , moyenne>

DF = {
 net → nom, âge, adresse
 matière → enseignant
 net, matière → moyenne
 âge → tarif }

CI référentielles :
 $\text{proj}_{\text{net}}(\text{NOTE}) \subseteq \text{proj}_{\text{net}}(\text{ETUDIANT})$
 $\text{proj}_{\text{matière}}(\text{NOTE}) \subseteq \text{proj}_{\text{matière}}(\text{ENSEIGNEMENT})$
 $\text{proj}_{\text{âge}}(\text{ETUDIANT}) \subseteq \text{proj}_{\text{âge}}(\text{PRIX})$

♦ Insertion dans PRIX :

On cherche à insérer du n-uplet n dans PRIX.

- pré-condition : $n.\text{âge} \notin \text{proj}_{\text{âge}}(\text{PRIX})$
- post-condition : $\text{PRIX} = \text{PRIX} \cup \{n\}$

♦ Suppression dans PRIX :

On cherche à supprimer un n-uplet n de PRIX.

- pré-condition : $n.\text{âge} \in \text{proj}_{\text{âge}}(\text{ETUDIANT})$
- post-condition : $\text{PRIX} = \text{PRIX} - \{n\}$

Suppression en cascade :

- pré-condition : VRAI
- post-condition :
 - $\text{PRIX} = \text{PRIX} - \{n\}$
 - $\text{ETUDIANT} = \text{ETUDIANT} - (\text{ETUDIANT} \mid \text{âge} = n.\text{âge})$
 - $\text{NOTE} = \text{NOTE} - (\text{NOTE} \mid \text{net} \in \text{proj}_{\text{net}}(\text{ETUDIANT} \mid \text{âge} = n.\text{âge}))$

♦ Insertion d'un n-uplet n dans ETUDIANT :

- pré-condition :
 - $n.\text{net} \notin \text{proj}_{\text{net}}(\text{ETUDIANT})$
 - $n.\text{âge} \in \text{proj}_{\text{âge}}(\text{PRIX})$

- post-condition : $ETUDIANT = ETUDIANT \cup \{n\}$
 - ♦ Suppression dans ETUDIANT :
 - pré-condition : $n.net \notin proj_{net}(NOTE)$
 - post-condition : $ETUDIANT = ETUDIANT - \{n\}$
- Suppression en cascade :
- pré-condition : VRAI
 - post-condition :
 - $ETUDIANT = ETUDIANT - \{n\}$
 - $NOTE = NOTE - (NOTE \mid net = n.net)$
- ♦ Insertion d'un n-uplet n dans NOTE :
 - pré-condition :
 - $(n.net, n.matière) \notin proj_{net, matière}(NOTE)$ (différent de $n.net \notin proj_{net}(NOTE)$ et $(n.matière) \notin proj_{matière}(NOTE)$)
 - $n.net \in proj_{net}(ETUDIANT)$
 - $n.matière \in proj_{matière}(ENSEIGNEMENT)$
 - post-condition : $NOTE = NOTE \cup \{n\}$
 - ♦ Suppression dans NOTE :
 - pré-condition : VRAI (ne vient pas de suppression en cascade mais de l'absence de conditions)
 - post-condition : $NOTE = NOTE - \{n\}$

Remarques :

L'ordre d'insertion / suppression dans les relations est important.

Vérifier qu'il n'y a pas de situation d'interblocage : par exemple insérer avant un net peut permettre d'insérer autre chose.

VI. SQL

A. Création de tables

Syntaxe :

CREATE TABLE <nom-de-table> (<colonne> | <contrainte-de-relation>)+

<nom-de-table> : chaîne de caractères simple (par exemple etudiant), ou composé (un nom de schéma suivi d'un nom de table, par exemple crous.etudiant).

<colonne> = <nom-de-colonne> <type-de-données> [<défaut>] [<contrainte-de-colonne>]

<type-de-données> :

- chaînes de caractères de longueur fixe : CHAR (size), où size est facultatif (omis : taille = 1, max = 255).
- chaînes de caractères de longueur variable :
 - VAR CHAR 2 (size), où size est obligatoire (max = 2000).
 - LONG, sans préciser de taille (on peut y mettre des valeurs jusqu'à 2 Go), mais avec restrictions :
 - un seul champ de type LONG par relation
 - contrainte d'intégrité : uniquement NULL ou NOT NULL pour ce champ (NULL = on ne peut rien y mettre)
 - index interdits
 - ne peut pas apparaître dans certaines parties d'une commande SQL (WHERE, GROUP BY, DISTINCT...)
- numérique (entier positif, négatif, virgule fixe ou flottante) : NUMBER(p,s), où :
 - p = précision : nombre total de chiffres ($1 \leq p \leq 38$), facultatif
 - s = scale : nombre de chiffres à droite de la virgule ($s > 0$), facultatif. Si $s < 0$, alors arrondi à gauche de la virgule.
 - NUMBER(p) : $s = 0 \rightarrow$ entiers.
 - NUMBER : $p = 38$ et s indéfini nombres à virgule flottante.

- par compatibilité, on peut utiliser NUMERIC, DECIMAL, INTEGER, INT, SMALLINT, FLOAT, REAL, DOUBLE PRECISION.
- dates : DATE, toutes les dates valides comprises entre le 1^{er} janvier 4712 av. JC et le 31 décembre 4712 ap. JC. Format : centenaire + année + mois + jour + heure + min + sec.

<défaut> : DEFAULT val, où val est la valeur que l'on donne à l'attribut lorsque l'utilisateur n'en fournit pas.
Restriction :

- cohérence de type
- ne peut pas contenir de référence à une autre colonne
- ne peut pas contenir de date non complètement spécifiée

<contrainte-de-colonne> : permet de spécifier différentes CI portant sur un seul attribut, y compris les contraintes référentielles :

- valeur null impossible : NOT NULL
- unicité de l'attribut : UNIQUE ou PRIMARY KEY
- contrainte référentielle : REFERENCES <table-référencée> [<colonne-référencée>] (le second argument est optionnel s'il est identique à la colonne référençante)
- contrainte générale : CHECK <condition>, la condition pouvant être exprimée avec :
 - <Attribut> θ <Valeur>, $\theta \in \{=, \neq, <, \leq, >, \geq\}$
 - prédicat d'intervalle : BETWEEN AND
 - prédicat de comparaison de texte : LIKE
 - test de nullité : NULL
 - test d'appartenance : IN

<contrainte-de-relation> : peuvent porter sur plusieurs attributs :

- contrainte d'unicité : UNIQUE (<attribut>)+
- contrainte référentielle : [FOREIGN KEY (<colonne-référençante>)+] REFERENCES <table-référencée> [(<colonne-référencée>)+]
- contrainte générale : CHECK <condition>

Exemple :

```
CREATE TABLE prix (
    age NUMBER (2) PRIMARY KEY,
    tarif CHAR (2) DEFAULT 'T1',
    CHECK (age > 25) )
```

```
CREATE TABLE etudiant (
    net NUMBER (2),
    nom VAR CHAR 2 (30),
    age NUMBER (2) [ REFERENCES prix [ age ] ],
    adresse VAR CHAR 2 (60),
    PRIMARY KEY (net),
    [ FOREIGN KEY (age) REFERENCES prix [ age ] ] )
```

B. Insertion de données

Syntaxe :

```
INSERT INTO <nom-de-table> [ ( <nom-de-colonne> )+ ]
    { VALUES ( <constantes> )+ | <commande-de-recherche> }
```

Exemple :

```
INSERT INTO etudiant (net, nom, age, adresse)
    VALUES (1, Meyer, 28, Strasbourg)
```

```
INSERT INTO prix
    VALUES SELECT DISTINCT age FROM etudiant WHERE...
```

Remarque : si toutes les colonnes ne sont pas précisées, les manquantes sont remplies avec NULL.

C. Suppression de données

Syntaxe :

```
DELETE FROM <nom-de-table>  
[ WHERE { <condition-de-recherche> | CURRENT OF <nom-de-curseur> } ]
```

Exemple :

```
DELETE FROM etudiant WHERE nom LIKE Meyer OR age > 36
```

```
DELETE FROM prix
```

Remarque : le second exemple élimine toute la table !

D. Mises à jour

Syntaxe :

```
UPDATE <nom-de-table>  
SET { <nom-de-colonne> = { <expression-de-valeur> | NULL } } +  
WHERE { <condition-de-recherche> | CURRENT OF <nom-de-curseur> }
```

Exemple :

```
UPDATE prix  
SET tarif = 'T12'  
WHERE age = 30
```

```
UPDATE etudiant \  
SET age = age + 1  
WHERE nom = Meyer
```

Remarque : seuls les n-uplets vérifiant la condition (optionnelle) sont mis à jour.

Chapitre 3 : Le modèle Entités / Associations

(source : C. Silber)

(E/A, ou E/R en anglais)

Modèle plus intuitif que le modèle relationnel, le modèle E/A est à la base de MERISE.

Concepts voisins, le passage de l'un à l'autre est immédiat.

Evolution du modèle E/A : il existe des modèles étendus :

- concepts de base : Entité, association, attribut
- concepts étendus : agrégation (abstraction), spécialisation (concrétion)

Amalgame de nombreux modèles, en gardant le meilleur de chacun.

Actuellement : UML

I. Entité

Entité = individu dans MERISE

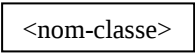
Entité = objet concret ou abstrait qui peut être distinctement identifié dans l'esprit du concepteur et qui présente un intérêt particulier.

Exemple : M. Meyer, département de vente d'une entreprise... → M. Meyer est UNE entité.

classe (ou type) d'entité : une classe regroupe toutes les entités de même nature.

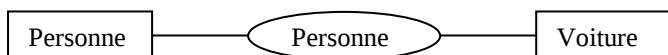
Exemple : M. Meyer → classe Personne (M. Meyer est une occurrence de la classe Personne).

Représentation graphique :



En général, une entité représente le nom dans une phrase.

Exemple : M. Meyer possède une 205 :



II. Association

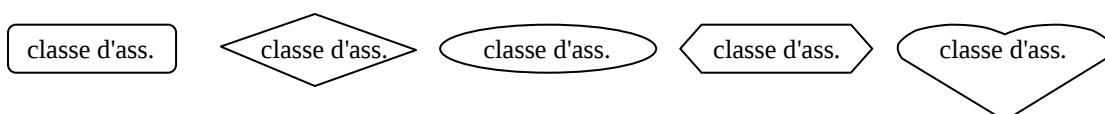
Association = relation dans MERISE.

Une association est une relation entre une ou plusieurs entités.

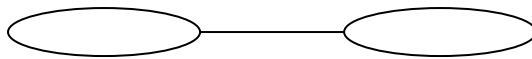
L'association représente généralement le verbe d'une phrase.

Classe (ou type) d'association = ensemble des associations ayant même sémantique, définies sur un même type d'entités.

Représentation graphique : il en existe de nombreuses, il faut en choisir une.

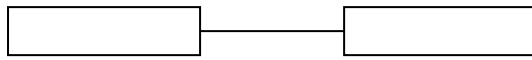


Un type d'association ne peut pas être directement relié avec un autre type d'association :



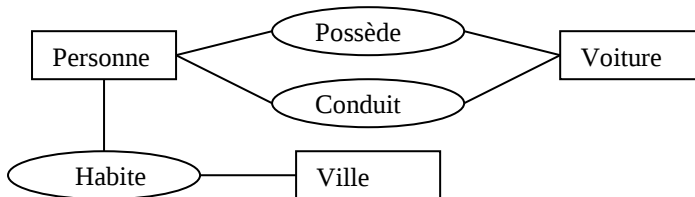
INTERDIT (sauf extension agrégation)

Un type d'entité ne peut être relié à un autre type d'entité que par un lien d'association :



INTERDIT

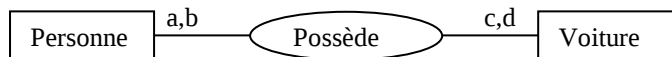
Il n'y a pas de restriction sur le nombre de types d'associations entre types d'entités :



Une occurrence d'association associe plusieurs entités : une et une seule entité par classe d'entités intervenant dans le type d'association.

III. Cardinalités et fonctionnalités

Cardinalité min et max reportées sur la graphique :



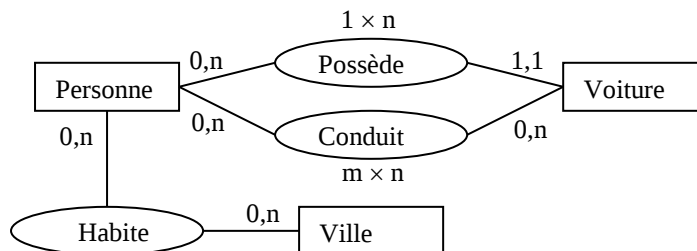
a = nombre min d'occurrences de B liées à l'occurrence de A par A - B ($\in \{0,1\}$)

b = nombre max d'occurrences de B liées à l'occurrence de A par A - B ($\in \{1,n\}$)

c = nombre min d'occurrences de A liées à l'occurrence de B par A - B ($\in \{0,1\}$)

d = nombre max d'occurrences de A liées à l'occurrence de B par A - B ($\in \{1,n\}$)

Exemple :



Remarque : selon les auteurs, les couples (a,b) et (c,d) peuvent être représentés sur une liaison ou sur l'autre.

♦ cardinalité : couple (a,b) ou (c,d). 4 alternatives pour les deux couples (a,b) et (c,d) :

- 0,1
- 0,n
- 1,1
- 1,n

Pas d'autre choix possible.

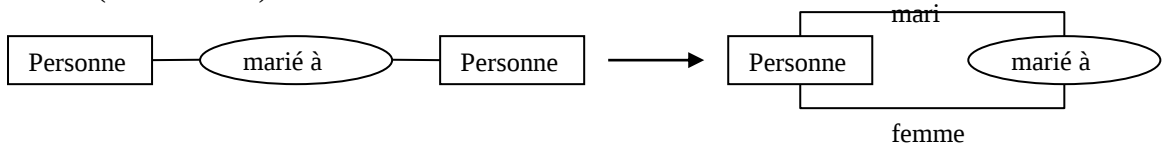
♦ fonctionnalité : couple (b,d). 3 alternatives :

- 1 x 1
- 1 x n
- n x m (= n x n car les deux n peuvent être différents, il s'agit du max des 2 cardinalités)

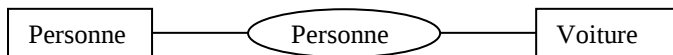
♦ Concept de dimension (ou ordre ou degré) du type d'association : c'est le nombre de types d'entités distinctes qui interviennent dans le type d'association.

Exemples :

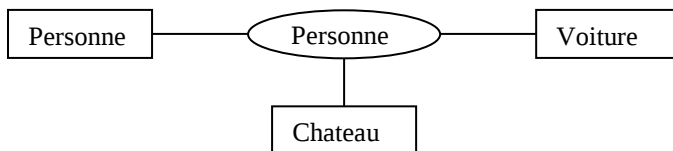
relation unaire (dimension = 1) :



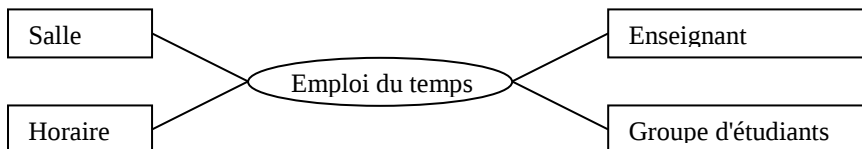
relation binaire (dimension = 2) :



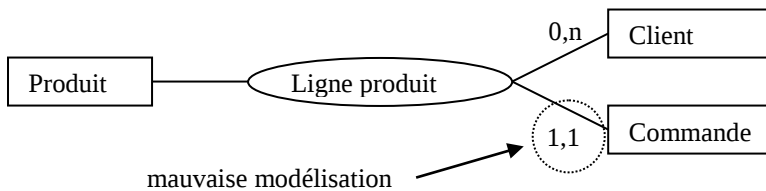
relation ternaire (dimension = 3) :



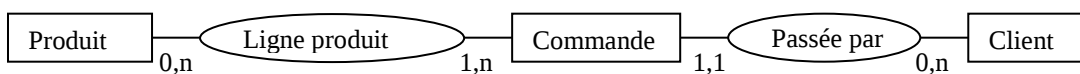
...relation n-aire (dimension = n) :



♦ Quand il y a plus de deux types d'entités, il peut y avoir des problèmes :



♦ Les cardinalités peuvent varier selon les questions que l'on se pose :



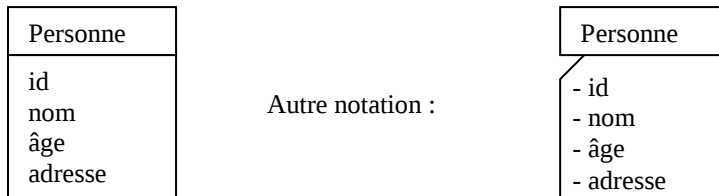
IV. Attribut

attribut = propriété dans MERISE

Un attribut est la propriété d'un type d'entité ou d'association.

Une propriété a une valeur.

Exemple :



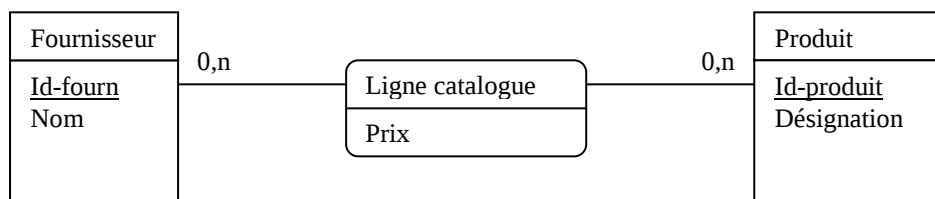
Attributs :

- valeurs simples, pas de structure, pas d'ensemble
- valeurs facultatives ou obligatoires

Les attributs peuvent être :

- des identifiants : clés, soulignés dans la représentation graphique
- des descripteurs : tous les autres.

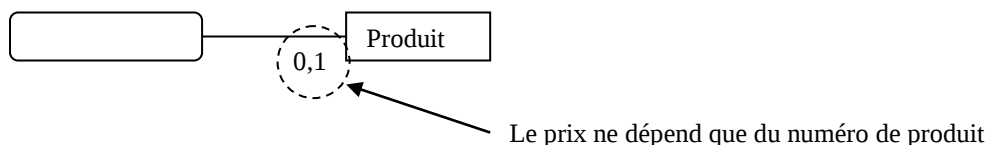
Les attributs du type d'association regroupent les clés de chacun des types d'entités de l'association + des attributs propres à l'association :



Clé de l'association : a priori, elle est constituée de toutes les clés de l'ensemble des clés des entités qui interviennent dans l'association :

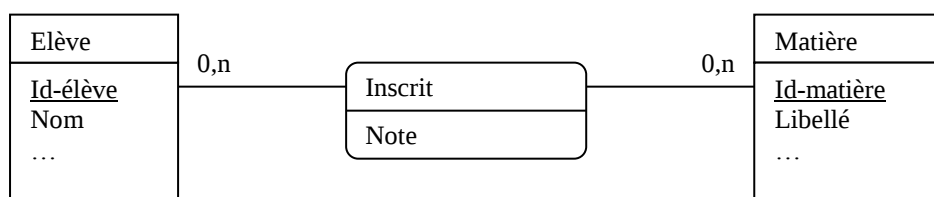
Exemple : la clé de ligne catalogue est (Id-fourn, Id-produit).

En pratique, ce n'est pas toujours le cas :



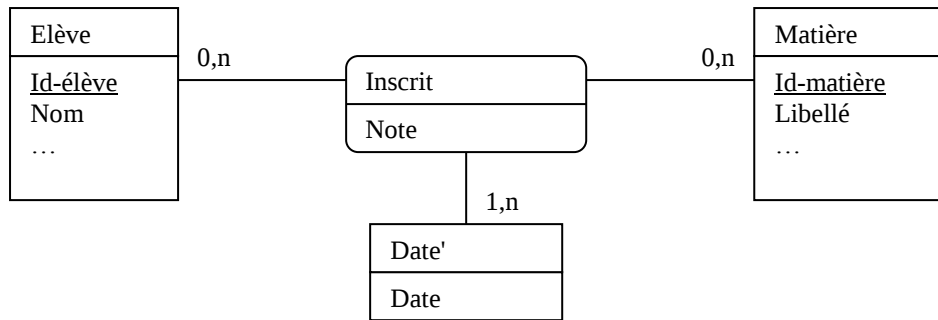
Dans ce cas, le prix peut se placer ailleurs (dans produit).

Exemple :



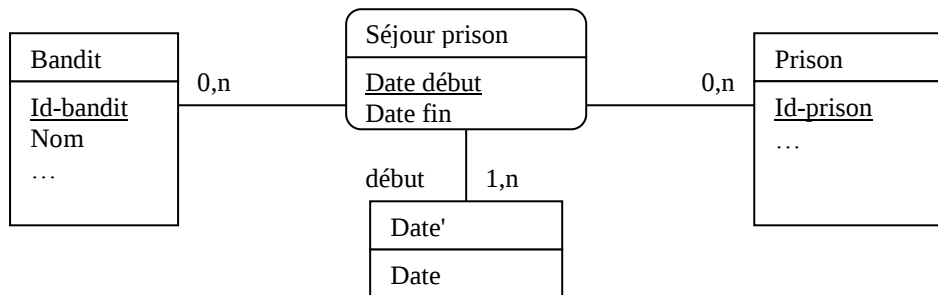
inscrit < Id-élève, Id-matière, note > n'autorise pas d'avoir plusieurs notes pour un étudiant donné dans une matière donnée.

Remède : analyser la cause de la répétitivité et introduire un type d'entités pour la traduire :



inscrit < Id-élève, Id-matière, Date, note >, on peut à présent avoir plusieurs notes par élève (une par date).

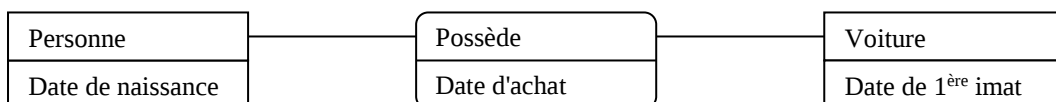
Il existe des exemples où cela ne fonctionne pas :



clé : id-bandit + id-prison + date + date-début → une date en entité + une date en association.

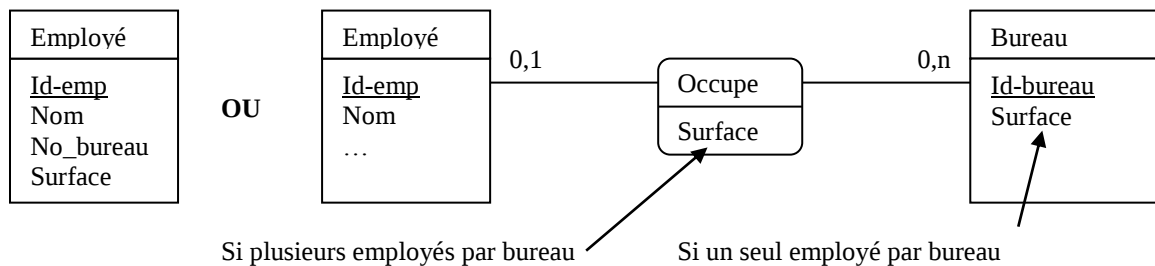
V. Règles de normalisation

- ♦ L'attribut est affecté au type d'entités ou d'associations qu'il décrit le plus directement :



Les trois dates sont très différentes.

- ♦ Choix entité / attribut :



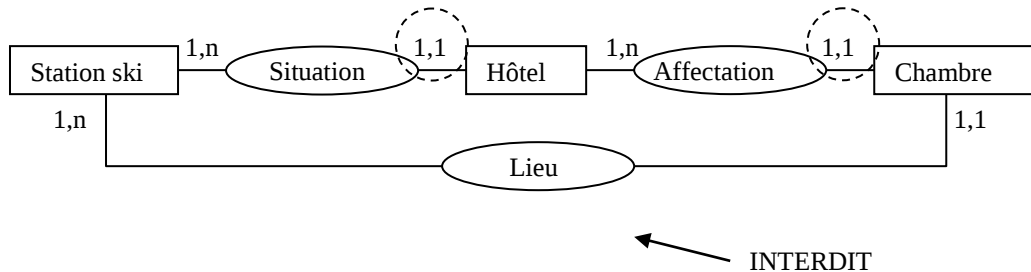
Le choix dépend de l'importance du bureau.

Un type d'entité qui ne comporte qu'un identifiant peut être traité comme un attribut, à condition qu'il n'intervienne pas dans un nouveau type d'association et qu'il ne génère pas d'attribut à valeur multiple.

♦ Il faut toujours vérifier si les associations de degré supérieur à deux ne peuvent pas être rapportées à plusieurs associations de degré inférieur.

♦ Un type d'association peut être éliminé s'il est la composition de deux types d'associations qui, par des cardinalités (0,1) ou (1,1), expriment la même réalité.

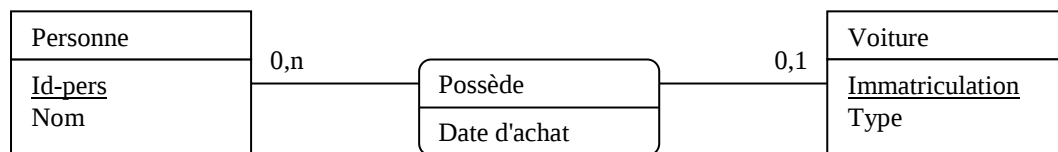
Exemple :



VI. Passage du modèle E/A au modèle relationnel

Une entité → une relation
 Une association → soit intégrée dans une relation existante (card max 1)
 soit donne lieu à une nouvelle relation

A. Exemple voitures



Plusieurs possibilités, selon les cardinalités :

Personne < Id-pers., nom, date, Immatriculation, type >

Personne < Id-pers., nom >

Voiture < Immatriculation, type >

Possède < Id-pers., Immatriculation, date >

Personne < Id-pers., nom >

Voiture < Immatriculation, type, Id-pers., date >

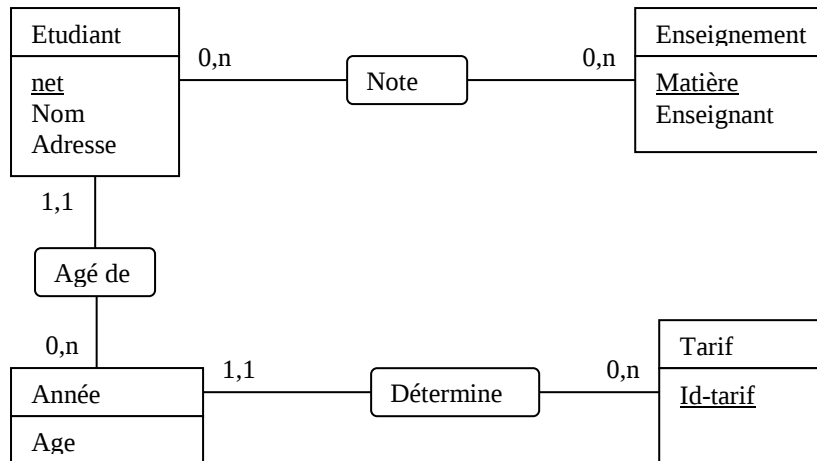
Contrainte d'intégrité référentielle pour le dernier cas :

$\text{proj}_{\text{Id-pers.}}(\text{Voiture}) \subset \text{proj}_{\text{Id-pers.}}(\text{Personne}) \cup \{ \text{NULL} \}$

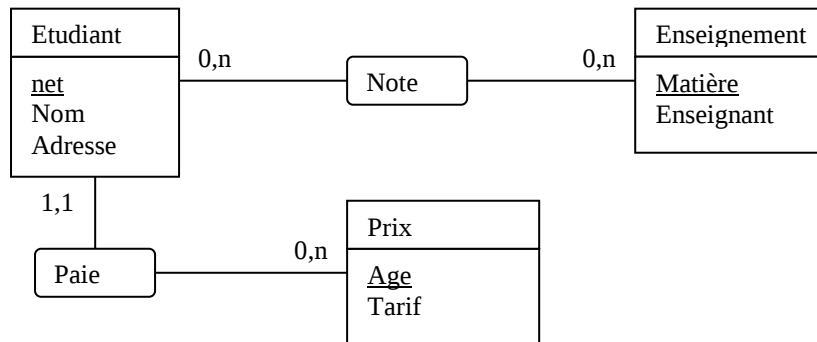
B. Exemple étudiant

ETUDIANT < net, nom, âge, adresse >
 PRIX < âge, tarif >
 ENSEIGNEMENT < matière, enseignant >
 NOTE < net, matière, moyenne >

Passage du modèle relationnel au modèle E/A :

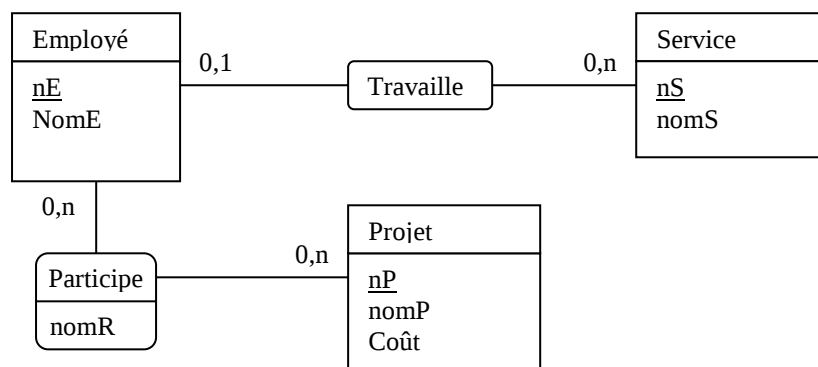


Ou encore :



C. Exemple Employé

SERVICE <nS, nomS>
 EMPLOYE <nE, nomE, nS>
 PROJET <nP, nomP, coût>
 PARTICIPE <nE, nP, nomR>

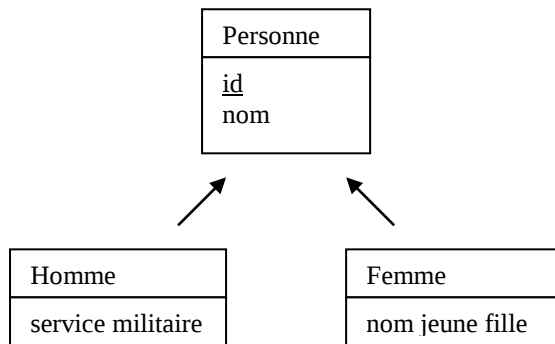


VII. Concepts complémentaires}

A. Généralisation / spécialisation

- ♦ Généralisation : regrouper les différents types d'entité en faisant abstraction de leurs différences.
→ type générique (mise en facteur des attributs communs).
- ♦ Spécialisation : pour un type donné, on définit des sous-types en mettant en évidence leur particularités.

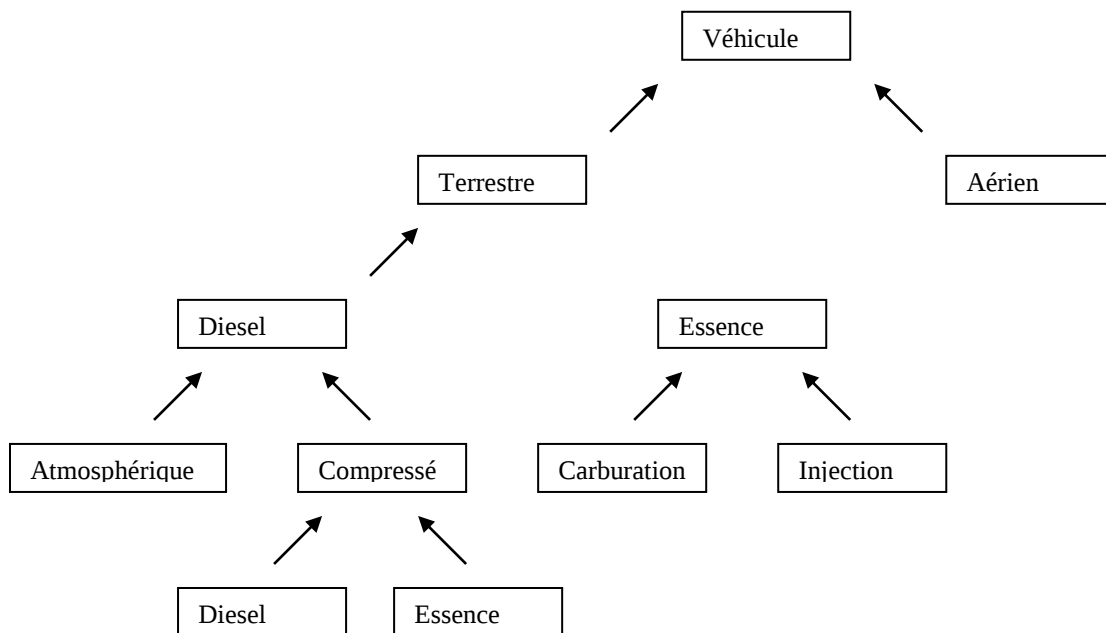
Exemple :



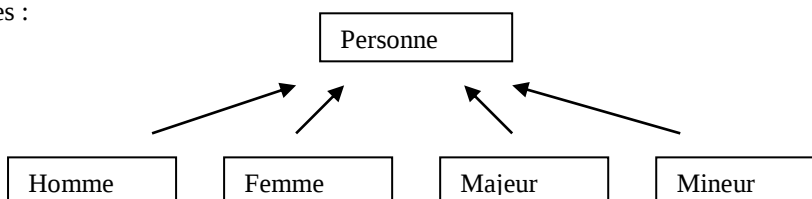
Il y a héritage des attributs du type générique :

Exemple : Homme < id, nom, service militaire > (id et nom proviennent de Personne).

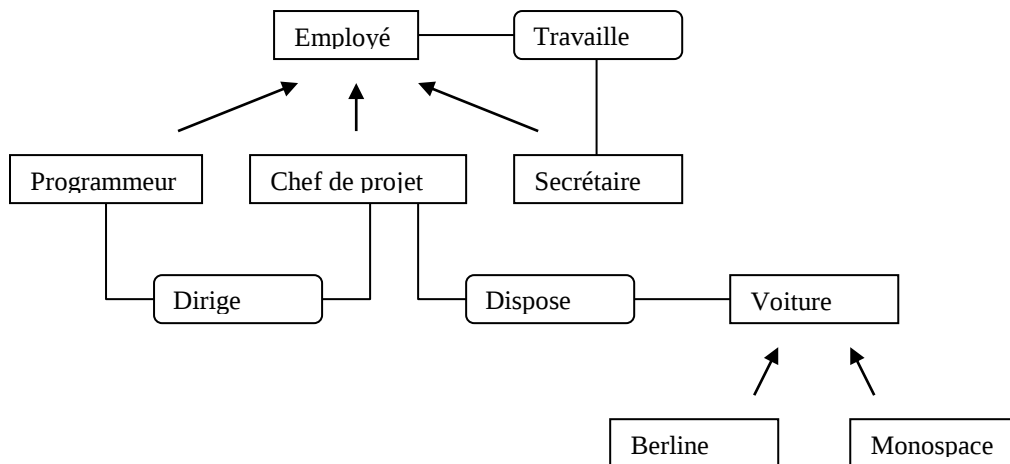
- ♦ Dans ce modèle, il n'y a plus de raison d'admettre des attributs à valeur nulle.
- ♦ Il peut y avoir N niveaux d'abstraction :



- ♦ Sur N critères :



♦ Association à tous les niveaux :



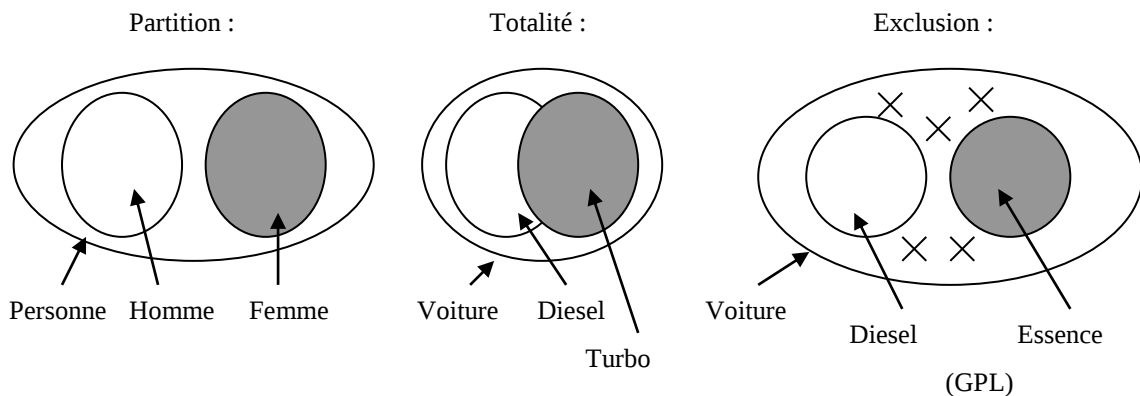
B. Modèle étendu : MERISE 2

Intégration d'informations supplémentaires dans le modèle E/A :

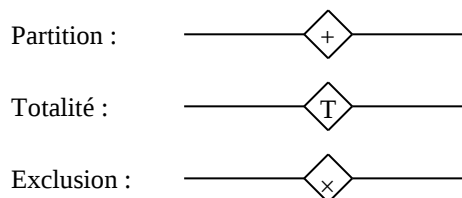
- partition (+) : disjonction + couverture (l'un ou l'autre)
- totalité (T) : $\neg$ disjonction + couverture (l'un ou l'autre ou les deux, mais pas autre chose)
- exclusion (×) : disjonction + $\neg$ couverture (l'un ou l'autre ou autre chose, mais pas les deux)

Fait appel aux notions de :

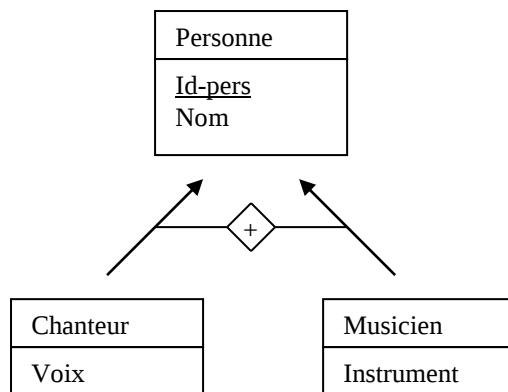
- Couverture : ensemble recouvrant tous les éléments.
- Disjonction : pas d'éléments commun à deux sous-ensembles.



Représentation graphique :



Exemple :



Expression des contraintes référentielles :

♦ Une relation :

personne <Id-pers, nom, voix, instrument >

partition : voix $\neq$ NULL XOR instrument $\neq$ NULL

totalité : voix $\neq$ NULL OR instrument $\neq$ NULL

exclusion : voix = NULL OU instrument = NULL

Autre possibilité :

personne < id-pers, nom, type, voix, instrument >

partition : type $\in$ {C,M}

type = C ET voix $\neq$ NULL ET instrument = NULL

type = M ET voix = NULL ET instrument $\neq$ NULL

totalité : type $\in$ {C,M,D}

exclusion : type $\in$ {C,M,NULL}

type = C ET voix $\neq$ NULL ET instrument = NULL

type = M ET voix = NULL ET instrument $\neq$ NULL

♦ Trois relations :

personne < id-pers, nom >
 chanteur < id-pers, voix >
 musicien < id-pers, instrument >

C1 : proj_{id-pers} (chanteur) $\subset$ proj_{id-pers} (personne)

C2 : proj_{id-pers} (musicien) $\subset$ proj_{id-pers} (personne)

C3 : proj_{id-pers} (musicien) $\cap$ proj_{id-pers} (chanteur) = $\emptyset$

C4 : proj_{id-pers} (musicien) $\cup$ proj_{id-pers} (chanteur) = proj_{id-pers} (personne)

partition : C3 (C1 et C2 non nécessaires)

totalité : C4 (C1 et C2 non nécessaires)

exclusion : C1 ET C2 ET C3

Autre méthode :

personne < id-pers, nom, type >
 chanteur < id-pers, voix >
 musicien < id-pers, instrument >

partition : $\text{type} \in \{C, M\}$
 $\text{proj}_{\text{id-pers}}(\text{chanteur}) \subset \text{proj}_{\text{id-pers}}(\text{personne} \mid \text{type} = C)$
 $\text{proj}_{\text{id-pers}}(\text{musicien}) \subset \text{proj}_{\text{id-pers}}(\text{personne} \mid \text{type} = M)$
totalité : $\text{type} \in \{C, M, D\}$
exclusion : $\text{type} \in \{C, M, \text{NULL}\}$
 $\text{proj}_{\text{id-pers}}(\text{chanteur}) \cap \text{proj}_{\text{id-pers}}(\text{personne} \mid \text{type} \in \{C, D\}) = \emptyset$
 $\text{proj}_{\text{id-pers}}(\text{musicien}) \cap \text{proj}_{\text{id-pers}}(\text{personne} \mid \text{type} \in \{M, D\}) = \emptyset$

VIII. Exercices de synthèse

A. Exercice employés.

1) Enoncé

Soit la base de données mémorisant les informations au sujet des employés des différents services d'une entreprise et leur participation aux projets. Chaque employé appartient à un seul service ; on ne mémorise pas l'historique de sa carrière. Il participe à un ensemble de projets avec différents rôles. Toutefois au sein d'un projet, un employé ne peut avoir plusieurs rôles.

Les attributs de la base sont :

nom de l'attribut	type	descriptif
nS	entier	numéro de service
nomS	chaîne	nom de service
nE	entier	numéro de l'employé
nomE	chaîne	nom de l'employé
nomR	chaîne	rôle de l'employé dans un projet
nP	entier	numéro de projet
nomP	chaîne	nom de projet
coût	réel	coût estimé d'un projet, éventuellement NULL

Les relations sont :

Service (nS, nomS) clé : nS
Employé (nE, nomE, nS) clé : nE
Projet (nP, nomP, coût) clé : nP
Participe (nE, nP, nomR) clé : (nE, nP)

avec les contraintes d'intégrité référentielles :

$\text{proj}_{\text{nS}}(\text{Employé}) \subseteq \text{proj}_{\text{nS}}(\text{Service})$
 $\text{proj}_{\text{nE}}(\text{Participe}) \subseteq \text{proj}_{\text{nE}}(\text{Employé})$
 $\text{proj}_{\text{nP}}(\text{Participe}) \subseteq \text{proj}_{\text{nP}}(\text{Projet})$

Donner les expressions en algèbre relationnelle et en SQL :

- Liste de tous les triplets (nP, nomP, coût) de tous les projets dont le coût est supérieur à 1M d'Euros.
- Liste de tous les triplets (nE, nomE, nS) des employés qui travaillent dans le service numéro 1, 5 ou 8.
- Liste de tous les rôles tenus par les employés.
- Liste de tous les numéros de projets.
- Liste de tous les noms et numéros des employés de la société.
- Nom des employés qui travaillent dans le service numéro 45.
- Numéros des employés qui travaillent pour le projet 72.
- Noms des employés qui travaillent pour le projet 98.

9. Noms et numéros de service des employés dont le rôle est "chef de projet" quel que soit le projet considéré.
10. Noms d'employés et numéros de service pour les employés travaillant dans un projet dont le nom est "station d'épuration".
11. Noms des employés travaillant dans un projet de coût inférieur à 1M d'Euros.
12. Numéros des employés qui ont le même rôle que celui occupé par l'employé 103 dans le projet 60.

2) Réponses

1. Liste de tous les triplets (nP, nomP, coût) de tous les projets dont le coût est supérieur à 1M d'Euros.

projet | coût $\geq$ 1M

```
SELECT * FROM Projet
      WHERE cout  $\geq$  1000000 ;
```

2. Liste de tous les triplets (nE, nomE, nS) des employés qui travaillent dans le service numéro 1, 5 ou 8.

Employé | nS $\in$ {1, 5, 8}

```
SELECT * FROM Employe
      WHERE nS = 1 OR nS = 5 OR nS = 8 ;
```

3. Liste de tous les rôles tenus par les employés.

proj_{nomR} (Participe)

```
SELECT DISTINCT nomR FROM Participe ;
```

4. Liste de tous les numéros de projets.

proj_{nomP} (Projet)

```
SELECT DISTINCT nomP FROM Projet ;
```

5. Liste de tous les noms et numéros des employés de la société.

proj_{nomE, nE} (Employé)

```
SELECT DISTINCT nomE, nE FROM Employe ;
```

6. Nom des employés qui travaillent dans le service numéro 45.

proj_{nomE} (Employé | nS = 45)

```
SELECT nomE FROM Employe
      WHERE nS = 45 ;
```

7. Numéros des employés qui travaillent pour le projet 72.

proj_{nE} (Participe | nP = 72)

```
SELECT nE FROM Participe
      WHERE nP = 72 ;
```

8. Noms des employés qui travaillent pour le projet 98.

proj_{nomE} (Employé $\times$ Participe | Employé.nE = Participe.nE ET np = 98)

ou

proj_{nomE} (Employe $\times$ (Participe | nP = 98)) | Employé.nE = Participe.nE

ou

proj_{nomE} (Employé | nE $\in$ proj_{nE} (Participe | nP = 98))

```
SELECT nomE FROM Employe, Participe
      WHERE Employe.nE = Participe.nE AND nP = 98 ;
```

ou

```
SELECT nomE FROM Employe
      WHERE nE IN (SELECT nE FROM Participe
```

WHERE nP = 98) ;

9. Noms et numéros de service des employés dont le rôle est "chef de projet" quel que soit le projet considéré.

proj_{nomE, nS} (Employé ∈ Participe | Employé.nE = Participe.nE ET nomR = chef de projet)

ou

proj_{nomE, nS} (Employé × Service × Participe | Employé.nS = Service.nS
ET Employé.nE = Participe.nE
ET nomR = chef de projet)

SELECT nomE, nS FROM Employe E, Service S, Participe P
WHERE E.nE = P.nE AND E.nS = S.nS AND nomR = 'chef de projet' ;

10. Noms d'employés et numéros de service pour les employés travaillant dans un projet dont le nom est "station d'épuration".

proj_{nomE, nS} (Employé × Participe × Projet | nomP = Station d'épuration
ET Projet.nP = Participe.nP
ET Participe.nE = Employé.nE)

SELECT nomE, nS FROM Employe E, Participe P, Projet J
WHERE nomP = Station d'épuration AND J.nP = P.nP AND P.nE = E.nE ;

11. Noms des employés travaillant dans un projet de coût inférieur à 1M d'euros.

proj_{nom} (Employé × Participe × Projet | coût ≤ 1M
ET Projet.nP = Participe.nP
ET Participe.nE = Employé.nE)

SELECT nomE FROM Employe E, Participe P, Projet J
WHERE cout ≤ 1000000 AND J.nP = P.nP AND P.nE = E.nE ;

12. Numéros des employés qui ont le même rôle que celui occupé par l'employé 103 dans le projet 60.

proj_{nE} (Participe | nomR ∈ proj_{nomR} (Participe | nE = 103 ET nP = 60))

ou

proj_{P2.nE} (Participe P1 ∈ Participe P2 | P1.nomR = P2.nomR
ET P1.nE = 103 ET P1.nP = 60)

SELECT P2.nE FROM Participe P1, Participe P2
WHERE P1.nomR = P2.nomR AND P1.nE = 103 AND P1.nP = 60 ;

B. Eléments de correction des examens de bases de données en DESS CCI précédents