Bases des données

Lamine BOUGUEROUA
Lamine.bougueroua@esigetel.fr

Plan

- Concepts de base
- Le modèle entité-association
- L’algèbre relationnel
- Le langage SQL
- Divers

Références

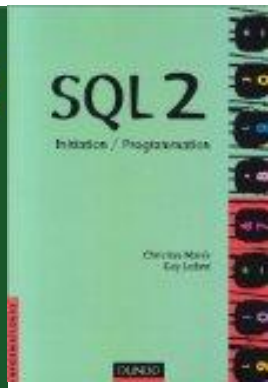
Références :

- Quelques ressources utiles:
  - http://sqlpro.developpez.com/cours/sqlaz/fondements
  - http://www.commentcamarche.net/merise
  - http://dev.ordi-facile.net/sql
  - http://www.ianywhere.com/developer/product_manuals/sqlanywhere/0901/fr/html/dbrffr9/00000062.htm
  - http://sqlpro.developpez.com/cours/sqlaz/ddl/?page=partie1#L0
  - http://www.ianywhere.com/developer/product_manuals/sqlanywhere/0902/fr/html/dbrffr9/dbrffr9.htm
  - http://cedric.cnam.fr/vertigo/

Références

Références :

- Quelques livres utiles:
  - Introduction aux bases de données par Chris J. Date, traduit par Frédéric Cuppens
  - SQL avancé . Joe Celko Thomson International Publishing.
  - Bases de données et SGBD, de la conception à la mise en oeuvre : démarche pratique. Kramarz Francis.
  - SQL2 Initiation / programmation par Christian Marée et Guy Ledant
  - Bases de données relationnelles : analyse et comparaison des systèmes. Gardarin et P. Valduriez
  - PostGreSQL par la pratique. John C. Worsley, Joshua D. Drake (O'Reilly – 2002 ISBN : 2-84177-211-X)



Partie I : Concepts de base

- Introduction générale
- Système de gestion (SGBD)
- Architecture
- Structures de données de base

Introduction générale

- Définition générale d'une base de données - BDD:

❑ Une BDD consiste en une collection de *données persistantes* utilisées par des systèmes d'application de certaines *entreprises*

✓ Entreprise : est un terme générique qui désigner toute entité commerciale, scientifique, technique ou autre organisation.
Ex. : banque, hôpital, université, ….

✓ Données persistantes : les données que doit maintenir une entreprise sur les opérations qu'elle réalise
Ex. : comptabilité, planification, production, ….

Introduction générale

- Historique

➢ Avant 1960 - organisation des documents en fichiers

➢ A partir du 1960 :
  - ✓ L'appariation des premiers gestionnaires des bases de données (SGBD)
  - ✓ Base de données hiérarchique *(programme Apollo - Nasa)*
  - ✓ Base de données réseaux *(Charles William Bachman)*

➢ 1970 – les premiers SGBD relationnels *(E.F. Codd)*

➢ 1980 – SGBD-O (orienté objet)

➢ 2000 – Base XML (hiérarchique)

➢ ….

Introduction générale

- Utilisation des BDD :

- BDD bibliographique
- BDD multimédia
- BDD chimique
- BDD géographique (navigation GSP 3D, urbanisme, musée,…)
- BDD décisionnelles (marketing, gestion commerciales, RH,…)
- ….

Introduction générale

- Glossaire :

- Une BDD ou BD (base de données) est un ensemble d'informations cohérentes, structurées et mémorisées sur un support

- Un SGBD (système de Gestion de Base de Données) est un logiciel qui propose des outils de gestion de la BDD : stockage, structuration, maintenance, mise à jour, consultation,…

- Une Banque de données est composée de : BDDs, SGBD et de services

- Un SI (Système d'informations) désigne l'ensemble des éléments qui participent aux cycles d'exploitation d'une informations (stockage, traitement, diffusion,…) au sein d'une organisation

Introduction générale

- Glossaire - suite :

- Une base de données est constituée de deux types d'information :

  - le dictionnaire de données qui décrit le contenu de la BD : comment sont structurées les données, quels sont les utilisateurs autorisés, quels sont leurs droits d'accès...

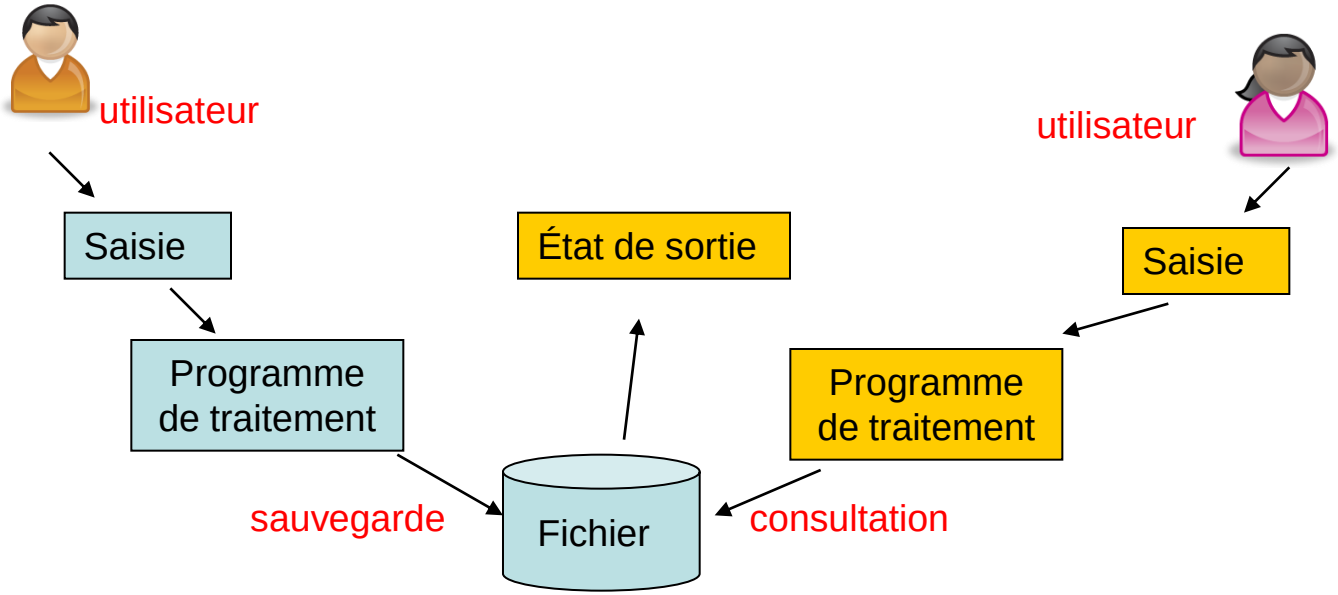
  - les données proprement dites. Par exemple, la description de chaque avion, les réservations...

Introduction générale

- Organisation des documents en fichiers :

➢ La première solution d'organisation des données

➢ Principe : saisie + traitement + sauvegarde dans un fichier
saisie + traitement + accès fichier + sortie



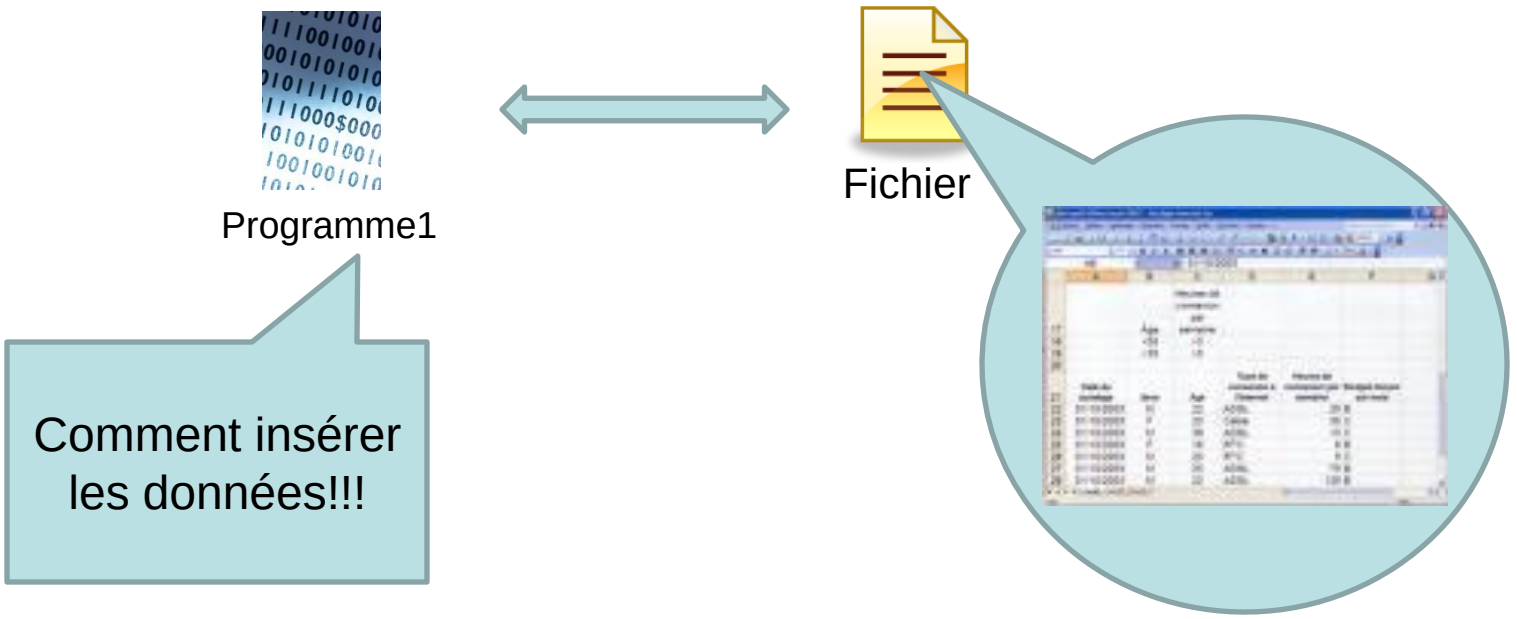
Introduction générale

- Exemple – agence de voyage:
- La gestion d'un vol :
  - Données : informations sur le vol (date de départ et d'arrivée, tarif,…)
  - Utilisateurs : les opérateurs
  - Sauvegarde : un opérateur peut réserver une place pour un client
  - Consultation : un opérateur peut obtenir des informations sur un vol
  - Mise à jour : un client peut demander des modifications sur les dates par exemple

Introduction générale

- Organisation des documents en fichiers – Limites :

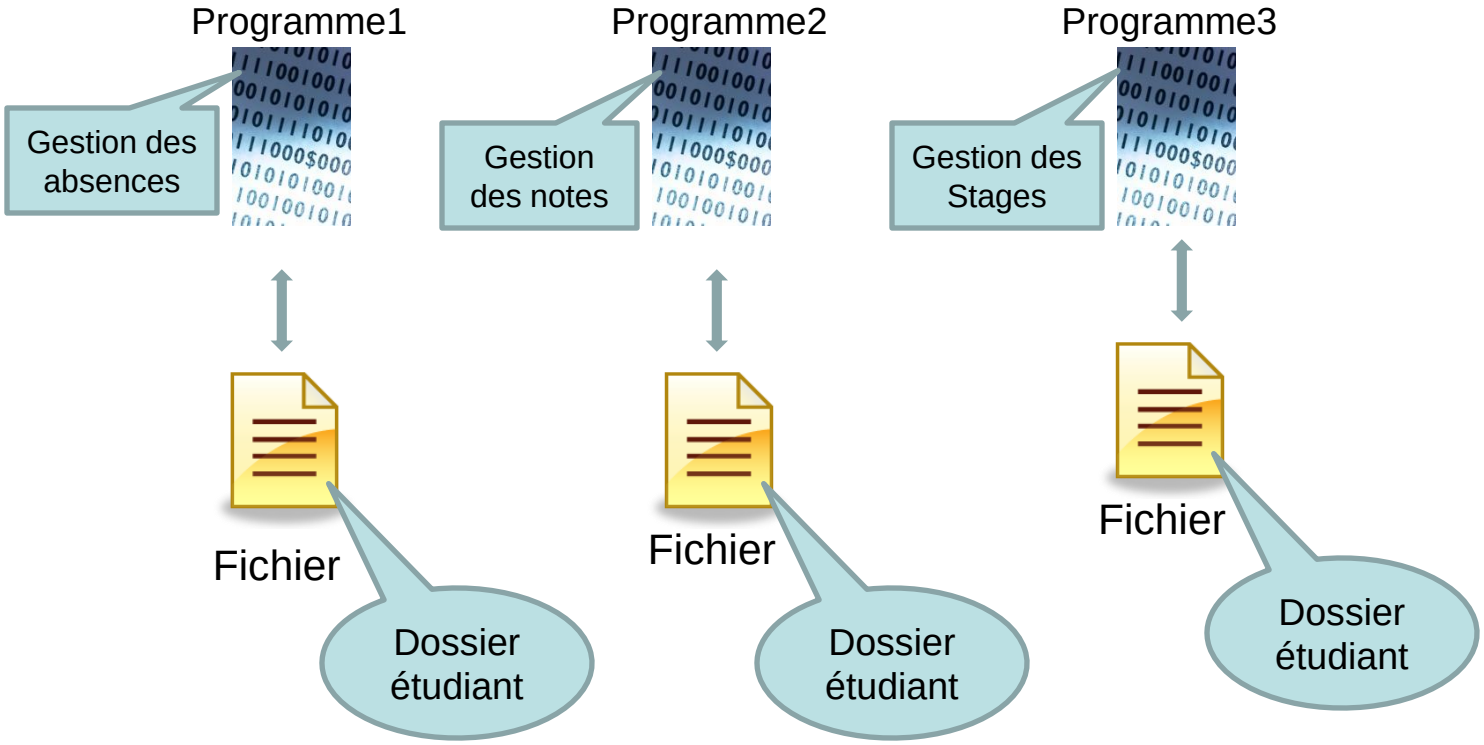
➢ Particularisation de la saisie et le traitement en fonction du fichier ➔ les programmes doivent connaître la description de la structure des fichiers



Introduction générale

- Organisation des documents en fichiers :

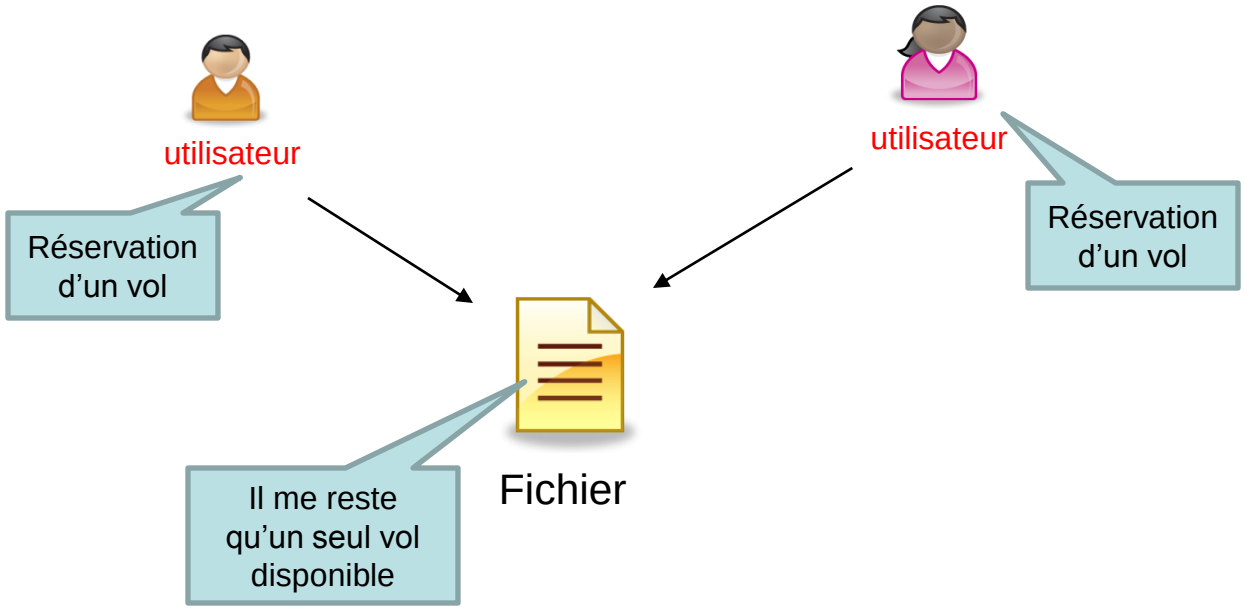
  - Particularisation des fichiers en fonction du traitement ➔ redondance des données importante



Introduction générale

- Organisation des documents en fichiers :

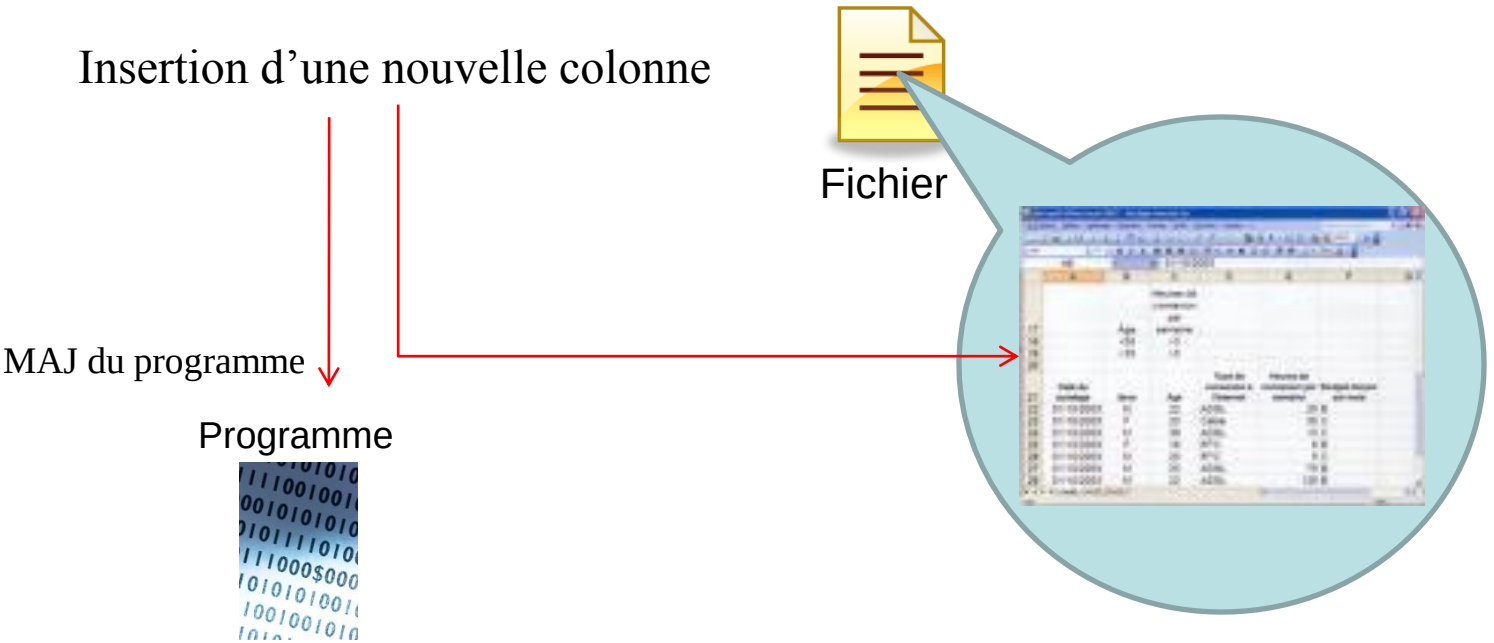
➢ Contrôle en différé des données ➔ risque d'erreurs et augmentation des délais



Introduction générale

- Organisation des documents en fichiers :

➢ Les programmes sont fortement liés aux structures de données ➔ l'évolution des applications est très difficile (nécessité de la réécriture des programmes)



Introduction générale

- Organisation des documents en fichiers :

  - Besoins d'uniformisation de la saisie et standardisation des traitements
  - Partage des données entre plusieurs traitements ➔ limitation de la redondance des données
  - Contrôle immédiat de la validité des données ➔ réduire les erreurs

- Nécessité d'un système de gestion des données => naissance des SGBD qui apportent une souplesse d'utilisation et de maintenance des données

Introduction générale

- Base de donnée – schéma d'organisation :

➢ L'avantage majeur par rapport à la première solution réside dans:

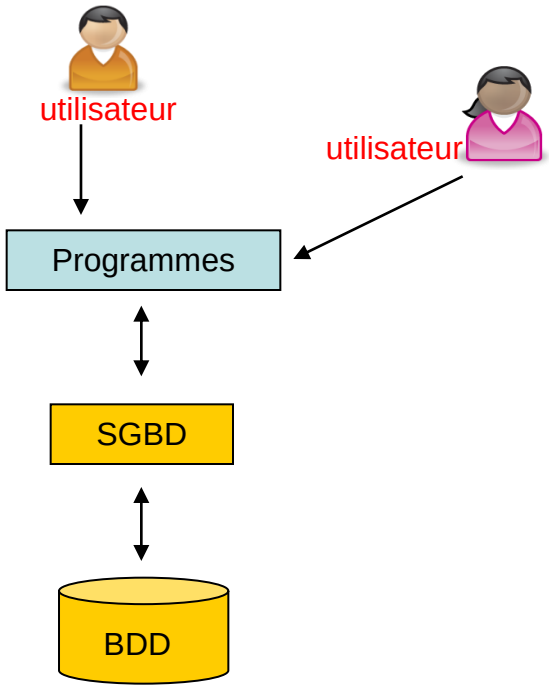
✓ L'indépendance entre les programmes et les données

✓ La centralisation de la gestion des données

✓ Usage multiple des données

✓ Evolution aisée

✓ Accès rapide, facile, protégé et souple



Introduction générale

- Base de donnée – Modèle de base données :

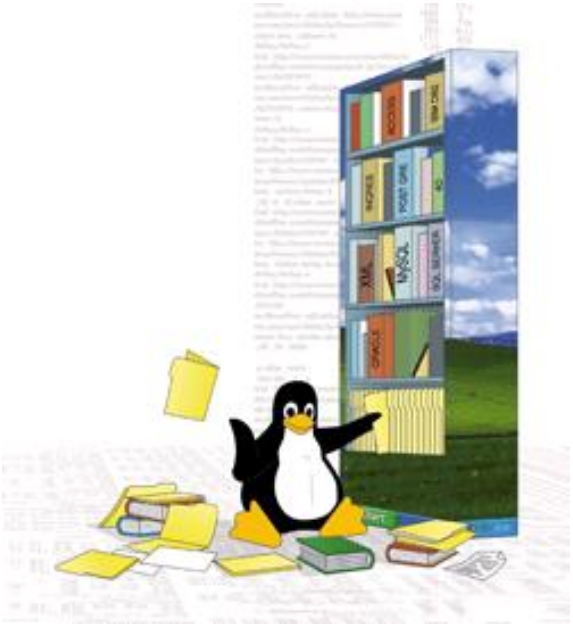
➢ Il existe plusieurs modèle de base de données :

✓ Modèle hiérarchique

✓ Modèle réseau

✓ Modèle relationnel

✓ Modèle objet



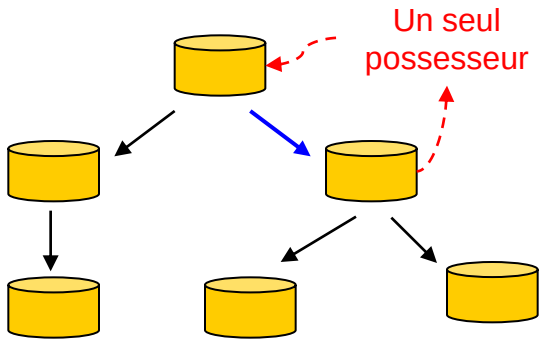
Introduction générale

- Modèle hiérarchique :

✓ Les données sont représentées sous forme d'une structure arborescente d'enregistrements

✓ Chaque enregistrement n'ait qu'un seul possesseur (ex. une paire de chaussures n'appartient qu'à une seule personne). Liaison de type *1-n*

✓ Cette structure est conçue avec des pointeurs qui déterminent le chemin d'accès au données



Introduction générale

- Modèle hiérarchique :

➢ Les structures de données hiérarchiques ont été largement utilisées dans les premiers systèmes de gestion de bases de données conçus pour la gestion des données du programme Apollo de la NASA (1965)

➢ Les liens hiérarchiques entre les différents types de données peuvent rendre très simple la réponse à certaines questions, mais très difficile la réponse à d'autres formes de questions

- Exemple : un étudiant peut avoir plusieurs enseignants et vis-vers-ca

➢ Nécessité de transformer la hiérarchie en réseau (modèle réseau)

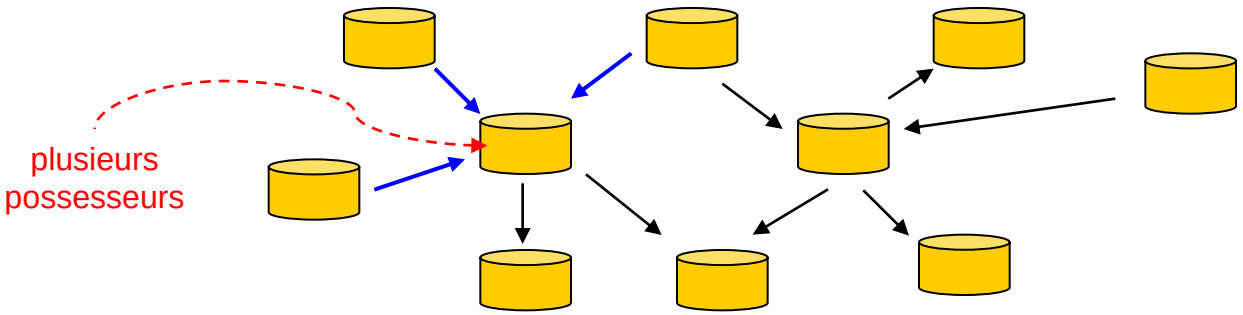
Introduction générale

- Modèle réseau :

✓ Ce modèle de bases de données a été inventé par *C.W. Bachman*. Pour son modèle, il reçut en 1973 le prix Turing (contribution de nature technique faite à la communauté)

✓ Le modèle réseau utilise des pointeurs vers des enregistrements comme le modèle hiérarchique

✓ La structure n'est plus forcément arborescente dans le sens descendant



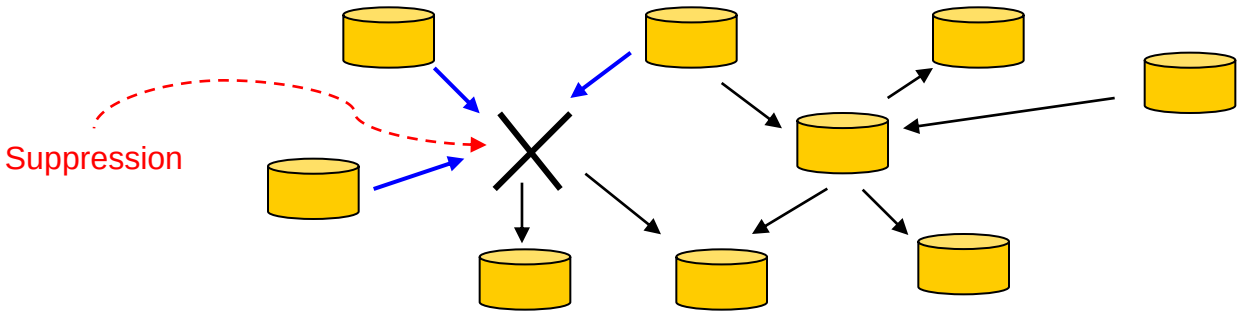
Introduction générale

- Modèle réseau :

✓ Possibilité d'établir des liaisons de type *n*-*n*, les liens entre objets pouvant exister sans restriction.

✓ Pour retrouver une donnée dans une telle modélisation, il faut connaître le chemin d'accès (les liens) ce qui rend les programmes dépendants de la structure de données

✓ Toute modification de la structure de donnée entraîne une reconfiguration profonde de la base de données



Introduction générale

- Modèle relationnel :

✓ Ce modèle est proposé par Edgar Frank Codd (Chercheur chez IBM) à la fin des année 1960

✓ L'objectif est de trouver de nouvelles méthodes pour gérer de grandes quantités de données

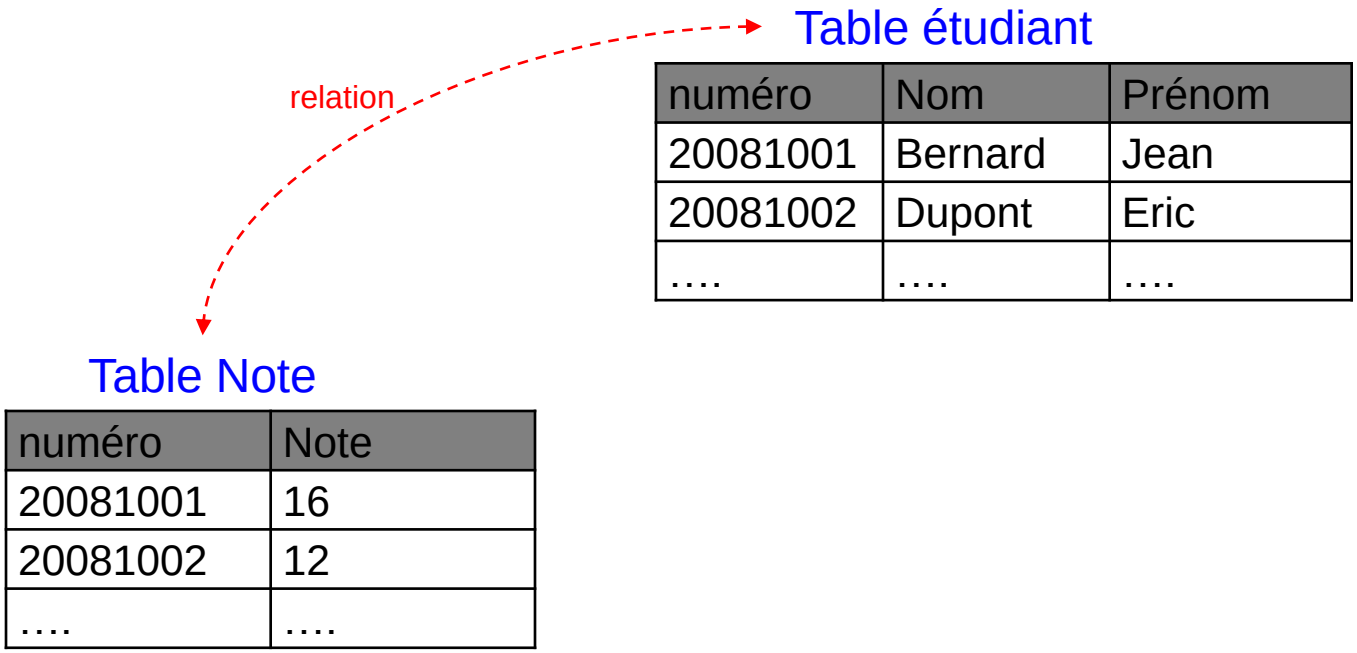
✓ Il proposait de stocker des données hétérogènes dans des tables, permettant d'établir des relations entre elles

✓ Actuellement, ce modèle est extrêmement répandu, mais en 1970, cette idée était considérée comme une curiosité intellectuelle

✓ C'est dans ce type de modèle que se situe ce cours de base de données.

Introduction générale

- Modèle relationnel - Exemple:



Introduction générale

- Modèle objet :

✓ Les données sont stockées sous forme d'objets, c'est-à-dire de structures appelées classes présentant des données membres

✓ Les champs sont des instances de ces classes

✓ Peu de bases de données 100% OO : migration (modélisation difficile) réservée aux "grandes" bases de donnée historique dans des domaines spécifiques: aéronautique, télécommunications, génome,…

Introduction générale

- Modèle XML :

✓ XML est un langage standard défini par le W3C : grâce à structure souple et générique, XML permet de représenter la plupart des structures de données

✓ C'est un langage permettant l'échange de donnée entre applications très différentes et d'être interrogé de façon uniforme (exemple: *XQuery*)

✓ XQuery, est un langage de requête, permet de considérer tout ensemble de documents XML comme une base de données sur laquelle les requêtes classiques sur les SGBD peuvent être appliquées

✓ De nombreux logiciels exportent leurs données sous forme XML, et de nombreux adaptateurs existent à présent, permettant de convertir en XML le format de données provenant de sources telles que des SGBD relationnelles, Objets, des documents textuels divers, des données provenant d'applications diverses ou de capteurs, des images, flux vidéo, …

Introduction générale

- Marché et stratégie :

✓ Les bases de données constituent aujourd'hui le cœur et la mémoire du système informatique d'une entreprise

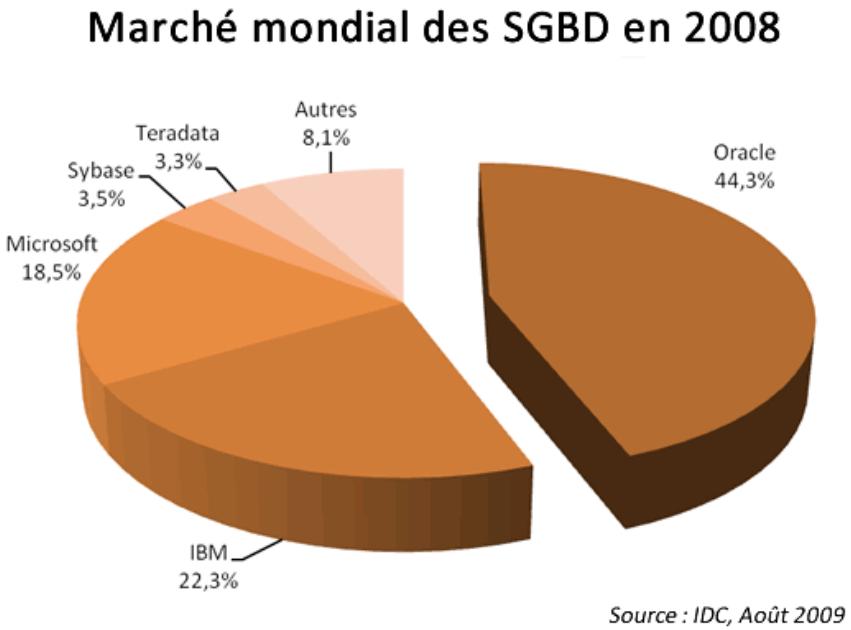
✓ Editeurs : Oracle (45%), IBM(22%), Microsoft, Sybase,…

Marché mondial des bases de données par chiffre d'affaires - en milliards de dollars - des éditeurs
Période : 2004 - 2006
Source : IDC 2007

	2004	2005	2006	2005 Share (%)	2005–2006 Growth (%)	2006 Share (%)
Oracle	6,003	6,376	7,312	44.3	14.7	44.4
IBM	2,923	3,113	3,483	21.6	11.9	21.2
Microsoft	2,013	2,442	3,052	17.0	25.0	18.6
Sybase	471	503	524	3.5	4.3	3.2
NCR Teradata	390	423	457	2.9	8.0	2.8
Other	1,495	1,542	1,624	10.7	5.3	9.9
Total	13,296	14,398	16,451	100.0	14.3	100.0

Introduction générale

- Marché et stratégie :



Introduction générale

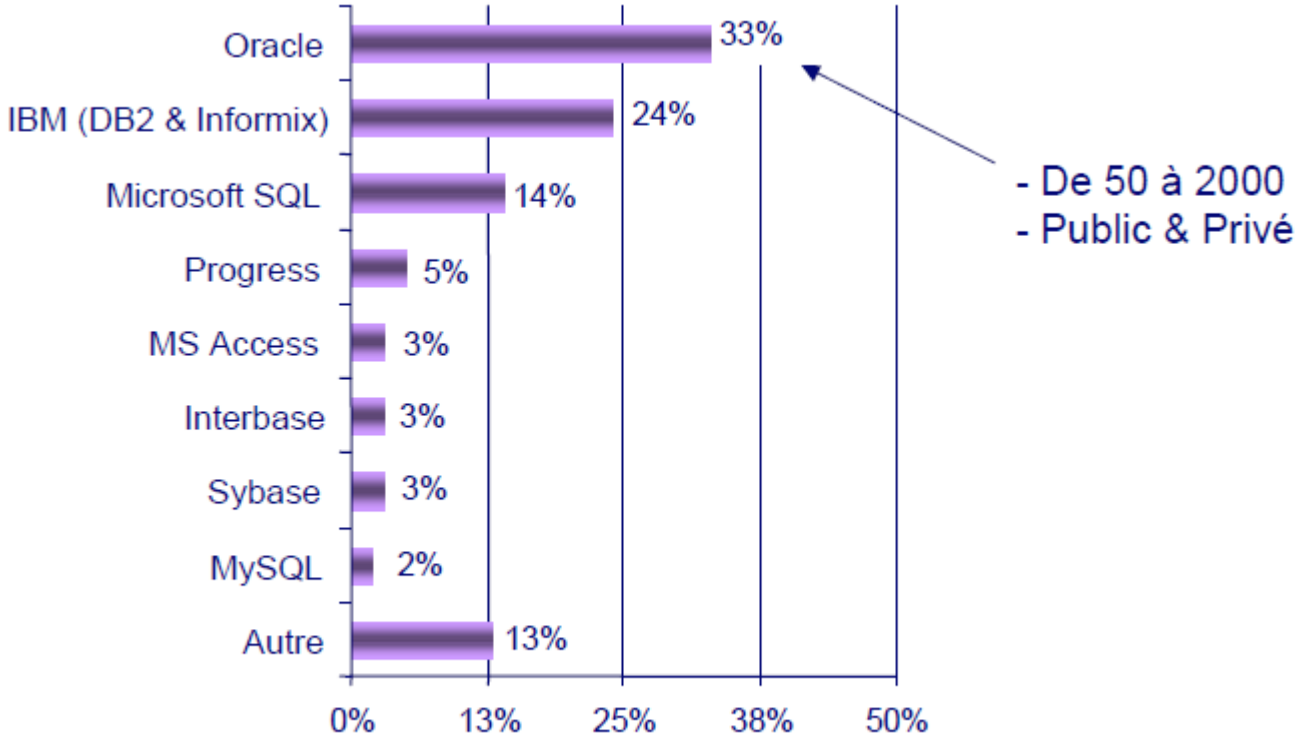
- Marché et stratégie :

- ✓ Une étude d'ITIC auprès de 450 entreprises fait apparaitre que 60% des entreprises prévoient une mise à jour de leurs bases de données en 2010 et 2011
- ✓ Les analystes soulignent notamment le rôle dans l'acquisition de Sun par Oracle en 2009 (Sun avait acheté MySQL en 2008)
- ✓ Le créateur de MySQL, Michael Widenius, a crée MariaDB en 2009
- ✓ Oracle, IBM, Microsoft et Sybase trustent 90% du marché
- ✓ Ces trois dernières années, seules 20% des entreprises ont changé de base de données. Dans 2 cas sur 3 c'était pour aller vers SQL Server
- ✓ Parmi les priorités des entreprises dans le choix de leur SGBD :
  - l'interopérabilité citée par 90% des répondants,
  - 80% le coût
  - 78% les performances.

Introduction générale

- Marché et stratégie :

✓ Les bases de données utilisées par les logiciels de gestion en place au sein des PME

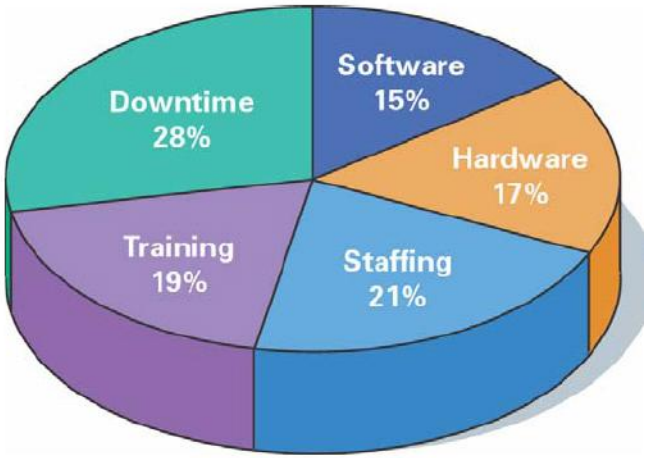


Source: IDC 2006

Introduction générale

- Marché et stratégie – bases de données :

✓ Les coût d'utilisation des bases de données



Downtime = Temps d'arrêt

Software = Licences

Hardware = Matériel

Staffing = Effectif

Training = Formation

Remarque : le temps d'arrêt d'un système est souvent le coût le plus important de n'importe quelle application (les entreprises ont mis en place des systèmes de service client en temps réel ou de ventes en ligne, tout temps d'arrêt a un impact direct sur les résultats)

Introduction générale

- Marché et stratégie – bases de données :

Bases de données gratuite ???

Introduction générale

- Marché et stratégie – open source:

✓ Les bases de données Open Source seront de plus en plus utilisées dans les projets d'entreprise, en premier lieu Ingres, PostgreSQL et MySQL

✓ Sachant que 80% des applications requièrent seulement 30% des fonctions proposées dans les principaux SGBD commerciaux, *Forrester* (cabinet d'étude) prédit un bel avenir à ces bases de données Open Source

Introduction générale

- Marché et stratégie – open source:

✓ Quelques exemples de bases de données Open Source :

Editeur	Site
PostgreSql	✓ http://www.postgresql.org/
InterBase / Firebird	✓ http://info.borland.com/devsupport/interbase/opensource/ ✓ http://firebird.sourceforge.ne
Ingres	✓ http://opensource.ca.com/projects/ingres

SGBD

- Propriétés d'organisation d'une base de données:

- Pourquoi un SGBD?

Coût réduit de stockage

Indépendance des données et des programmes

Accès facile

Disponibilité

Accès protégé

Souplesse

Accès rapide

Durabilité

Evolutive

Exactitude

Cohérence

Maintenabilité

SGBD

- SGBD - objectifs:

➢ Indépendance physique : les données sont définies d'une manière indépendante des structures de stockage utilisées

➔ un remaniement de l'organisation physique des données ne conduit pas à des modifications dans leur traitements (applications)

➢ Indépendance logique : un même ensemble de données peut être vu différemment par des utilisateurs différents

➔ un remaniement de l'organisation logique des données (ex. modification d'une rubrique) ne conduit pas à des modifications des applications non concernées

SGBD

SGBD - objectifs:

- Efficacité d'accès aux données : l'accès aux données se fait par l'intermédiaire d'un Langage de Manipulation de Données (LMD). Il est crucial que ce langage permette d'obtenir des réponses aux requêtes en un temps « raisonnable »

  ➔ garantie de bande passante (nb de transactions exécutées par seconde) et d'un bon temps de réponse (temps d'attente moyen pour une transaction)

- Administration facile des données : toutes les données doivent être centralisées dans un réservoir unique commun à toutes les applications

  ➔ les données sont administrées de façon centralisée

  ➔ outils spécialisés pour décrire et suivre l'évolution des données selon des habilitations

SGBD

SGBD - objectifs:

- Non redondance des données : pour éviter les problèmes lors des mises à jour, chaque donnée ne doit être présente qu'une seule fois dans la base

  ➔ diminution du volume de stockage

- Cohérence des données : vérification automatiquement des données à chaque insertion, modification ou suppression (ex. une date doit être valide)

  ➔ diminution des erreurs

- Partage des données : Il s'agit de permettre à plusieurs utilisateurs d'accéder aux mêmes données au même moment de manière transparente

  ➔ utilisation simultanée des données

SGBD

- SGBD - objectifs:

➢ Sécurité des données : les données doivent pouvoir être protégées contre les accès non autorisés

➔ droits d'accès aux données (comptes utilisateurs)

➢ Résistance aux pannes : donner la possibilité de pouvoir récupérer une base de données dans un état correct après une panne pendant une opération

➔ récupérer les données dans l'état dans lequel elles étaient avant l'opération ou bien terminer l'opération interrompue

SGBD

SGBD – fonctions :

- Description des données : Langage de Définition de Données (LDD)
- Manipulation des données : Langage de Manipulation de données (LMD)
  - ✓ Recherche
  - ✓ Mise à jour
- Contrôle des données :
  - ✓ Vérification de l'intégrité des données
  - ✓ Gestion des transactions (droits, …)

SGBD

- SGBD – résumé :

  - Une interface entre utilisateur et les informations (données) facilitant le travail des utilisateurs, donnant l'impression que l'utilisateur est seul à utiliser l'information

Les SGBD sont des composants fondamentaux de tout système d'information. Ils permettent de stocker et de récupérer des données.
Associées à des outils, ces données se transforment en informations, indispensables au fonctionnement des entreprises modernes

BDD

- Utilisateurs d'une base de données:

➢ Il existe plusieurs types d'utilisateurs d'une base de données:

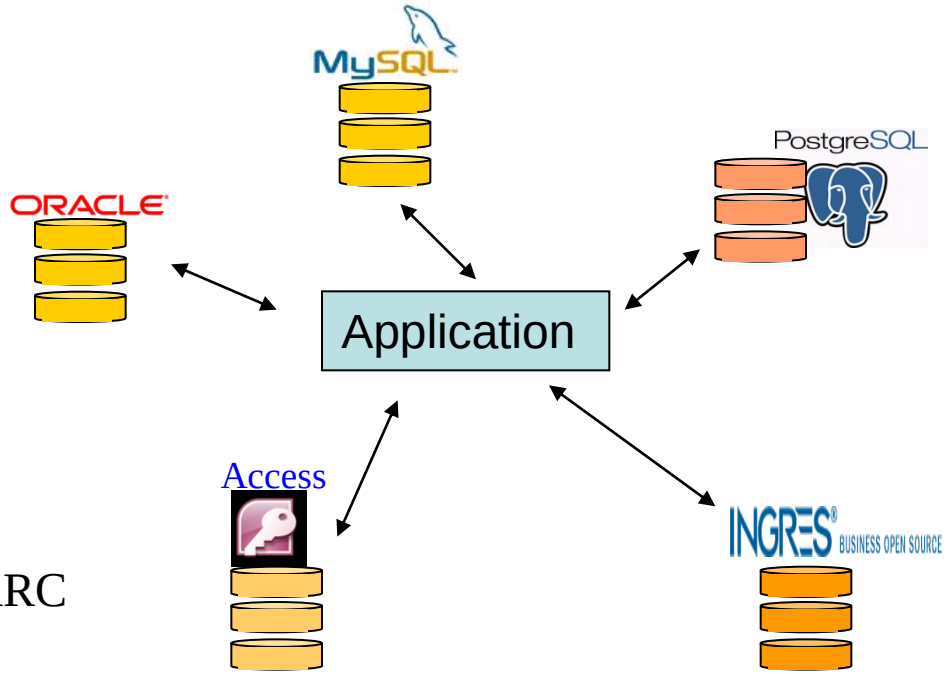
✓ Administrateur de la BDD (*dba*):
- Il est chargé du contrôle de la base de données (l'accès, maintenance,…)
- Il est chargé des sauvegardes et des procédures de reprise après une panne.

✓ Développeur d'application sur une BDD (*programmeur*) :
- Il développe des interfaces pour l'utilisation de la base de données,

✓ Utilisateur :
- Cherchent les infos, sans connaître la BD
- Il n'a accès qu'aux données qui lui sont utiles (consultation, modification, suppression, etc..)

Interopérabilité

- Interopérabilité est la capacité à rendre compatibles deux systèmes quelconques. L'interopérabilité nécessite que les informations nécessaires à sa mise en œuvre soient disponibles sous la forme de *standards ouverts*



- Nécessité de standards
  - Architecture: ANSI/SPARC
  - Modèles
  - Langage de requête

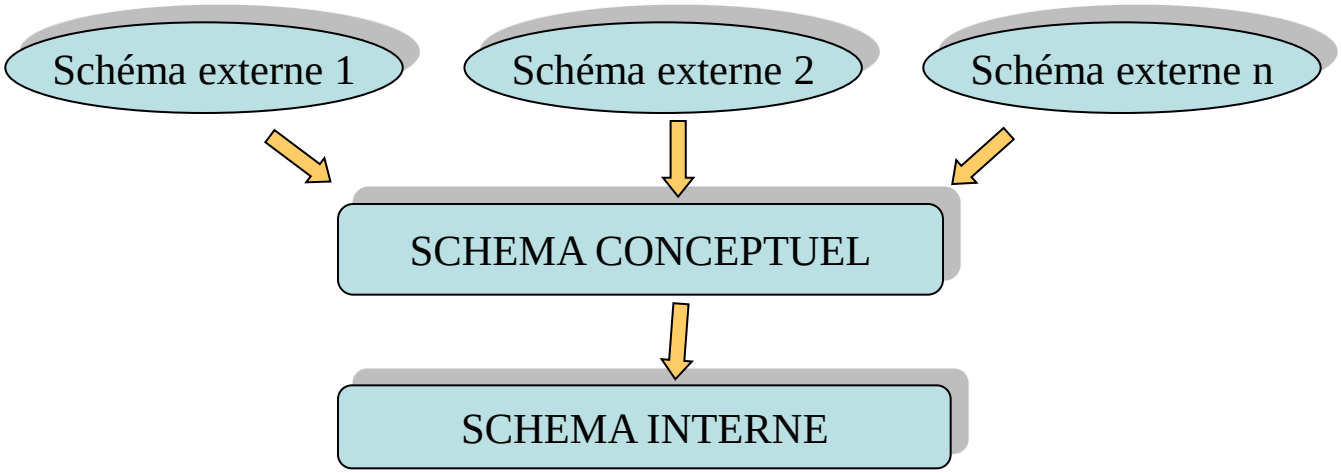
Architecture

- Pour atteindre certains de ces objectifs, trois niveaux de description des données ont été définis par la norme ANSI/SPARC :

✓ Le niveau externe

✓ Le niveau conceptuel

✓ Le niveau interne ou physique



Architecture

- Le niveau externe correspond aux différentes façons (vues) dont les utilisateurs peuvent voir les données de la base :

✓ On appelle cette description le schéma externe ou vue

✓ Il peut exister plusieurs schémas externes représentant différentes vues sur la base de données

✓ Le niveau externe assure l'analyse et l'interprétation des requêtes en primitives de plus bas niveau

✓ Il se charge également de convertir éventuellement les données brutes, issues de la réponse à la requête, dans un format souhaité par l'utilisateur

Architecture

- Le niveau conceptuel correspond à une vision globale de la base :

✓ On appelle cette description le schéma conceptuel (il n'y a qu'un seul schéma conceptuel)

✓ Les données sont décrites ainsi que les contraintes d'intégrité liées au données

✓ Ce niveau est appelé aussi modèle Entité-Relation (modèle conceptuel de données) ou bien MLD (modèle logique de données)

✓ Il décrit la structure de toutes les données de la base : leurs propriétés (i.e. les relations qui existent entre elles), sans se soucier de l'implémentation physique ni de la façon dont chaque groupe de travail voudra s'en servir

Architecture

- Le niveau conceptuel - exemples :

  - Type d'objets qui compose la BDD:

    Buveur(Nom, Prénom, Adresse)

    Vin(Cru, Millésime, Qualité, Quantité)

    Type d'associations:

    Abus(Buveur, Vin, Date, Quantité)

  - Type d'objets:

    Cinéma (Nom, Adresse)

    Film (Titre, Année)

    Type d'associations:

    Séance (Cinéma, Film, Horaire)

Architecture

- Le niveau interne ou physique correspond à la manière dont la base est implantée (fichier, table, support physique) et les méthodes pour y accéder

- Il s'appuie sur un système de gestion de fichiers pour définir la politique de stockage ainsi que le placement des données

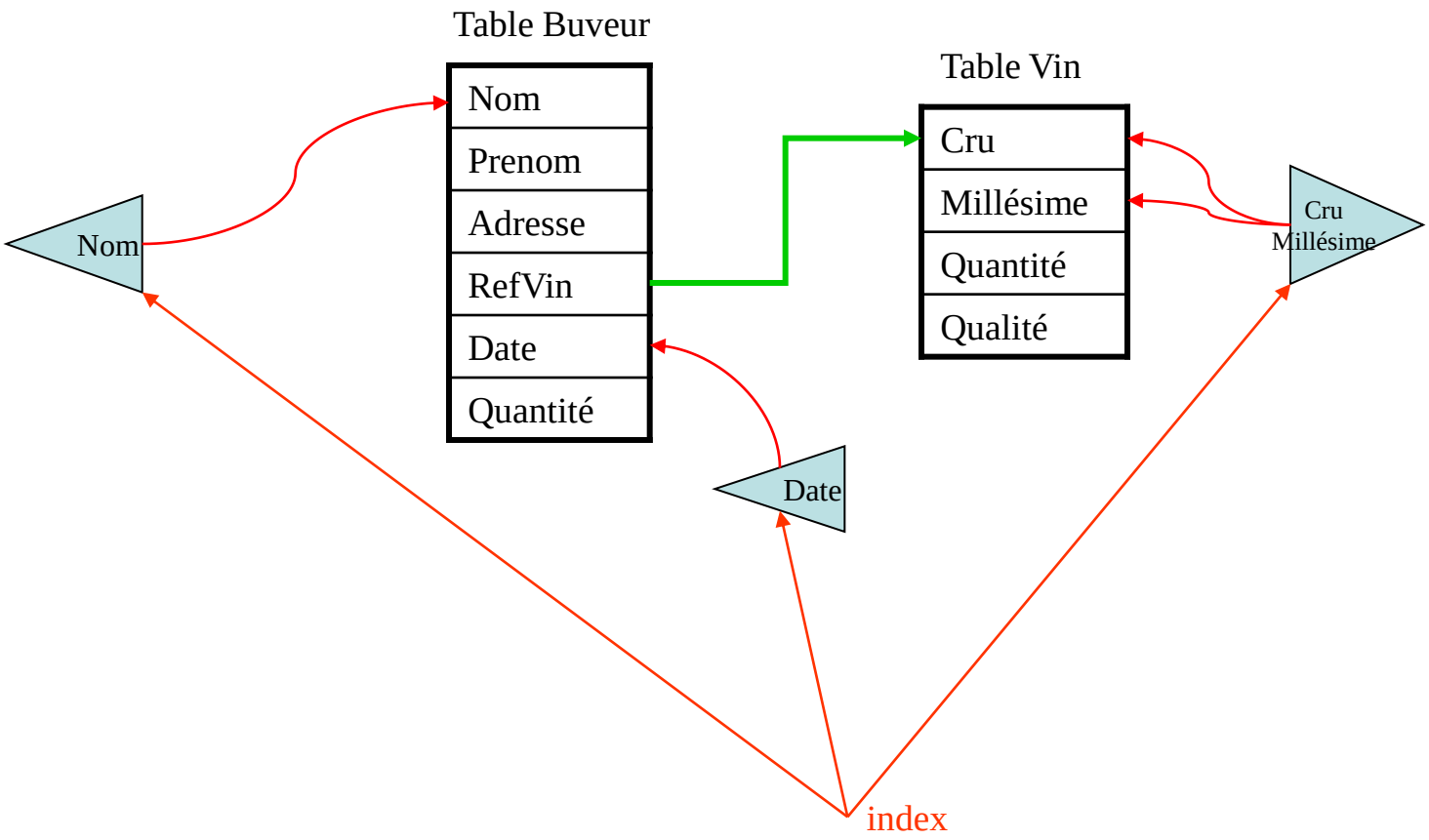
- Le niveau physique est donc responsable du choix de l'organisation physique des fichiers ainsi que de l'utilisation de telle ou telle méthode d'accès en fonction de la requête

- On appelle cette description le schéma interne

- Il dépend du SGBD

Architecture

- Le niveau interne - exemple :



Durée de vie d'une BDD

- La durée de vie d'une base de données:

➢ Une BDD passe par plusieurs étapes:

✓ Conception
  - Définition de la structure de la base

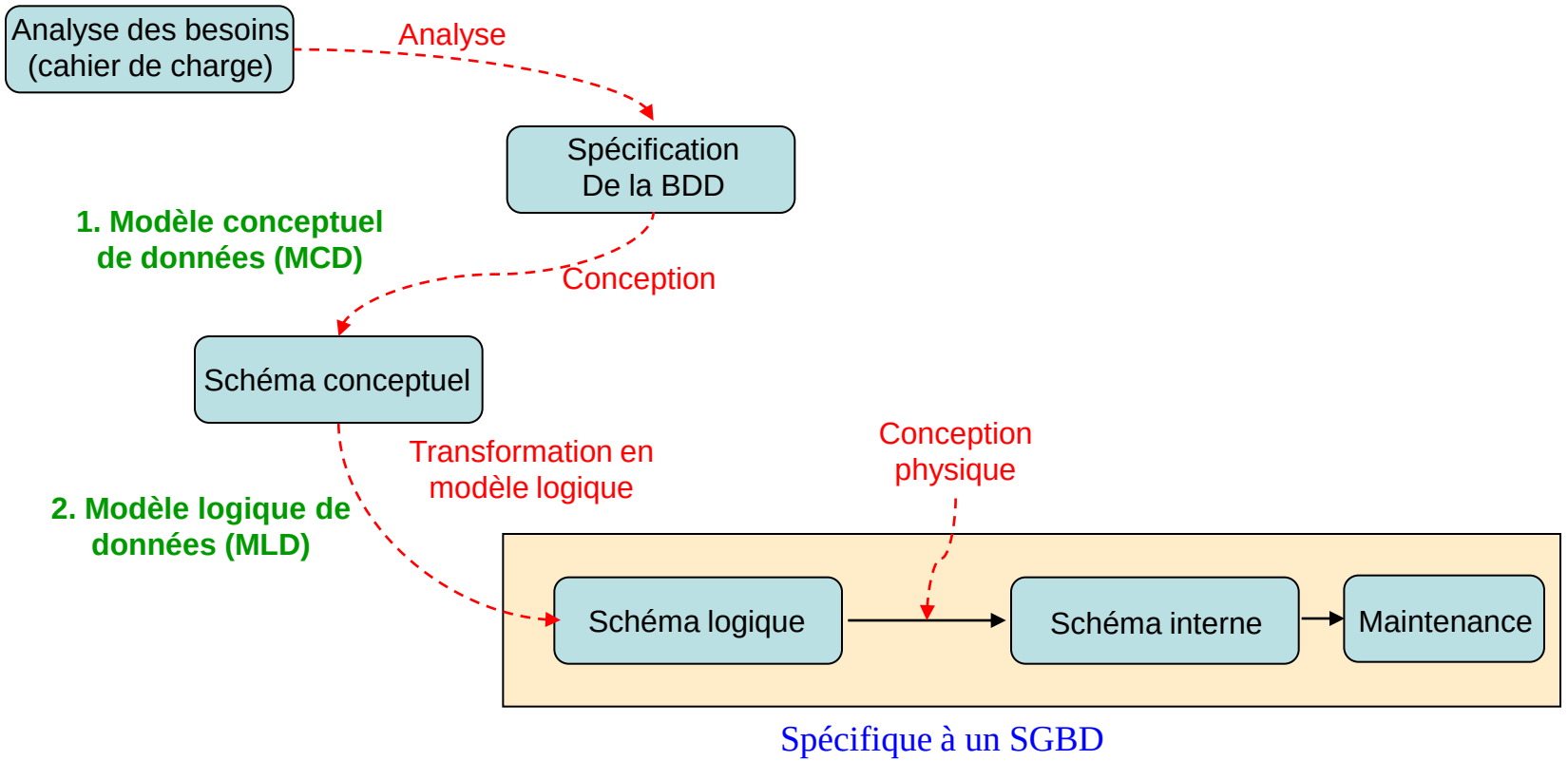
✓ Mise en place (*implantation*) :
  - Création de la BDD
  - Insertion des données

✓ Utilisation :
  - Interrogation de la base
  - Mise à jour des informations

✓ Maintenance :
  - Veille au bon fonctionnement de la base (correction et évolution)

Conception d'une BDD

- Conception d'une base de données :



Partie 2 : Modèle entités-associations (Modèle Conceptuel des Données – MCD)

- Introduction
- Entités
- Associations
- Cardinalité
- Méthodologie de construction d'un MCD
- Exemple
- Conseils de modélisation
- Normalisation

MCD : Introduction

- Modèle entités-association (Modèle E/A):

  - Appelé aussi : modèle conceptuel de données (MCD) ou Modèle Entité-Relation

  - Proposé par Chen (1976) et standardisé par l'ANSI en 1988

  - Il fournit une description graphique pour représenter de tels modèles de données sous la forme de diagrammes entité-relation

  - Ce modèle est utilisé dans les phases amont de conception des systèmes informatiques

MCD : Introduction

- Modèle entités-association - objectifs:

Représentation structurée des données:
indépendamment de l'implantation physique de SI

Documentation

Communication :
entre les différents acteurs

Dictionnaire des données:
le référentiel de l'entreprise ou du projet

MCD : Introduction

- Modèle entités-associations - principe:

➢ Le modèle entités-associations distingue les entités (objets) et les associations entre les entités (relations) :

- Une entité est similaire à la notion d'objet, elle décrit une «entité» du monde réel

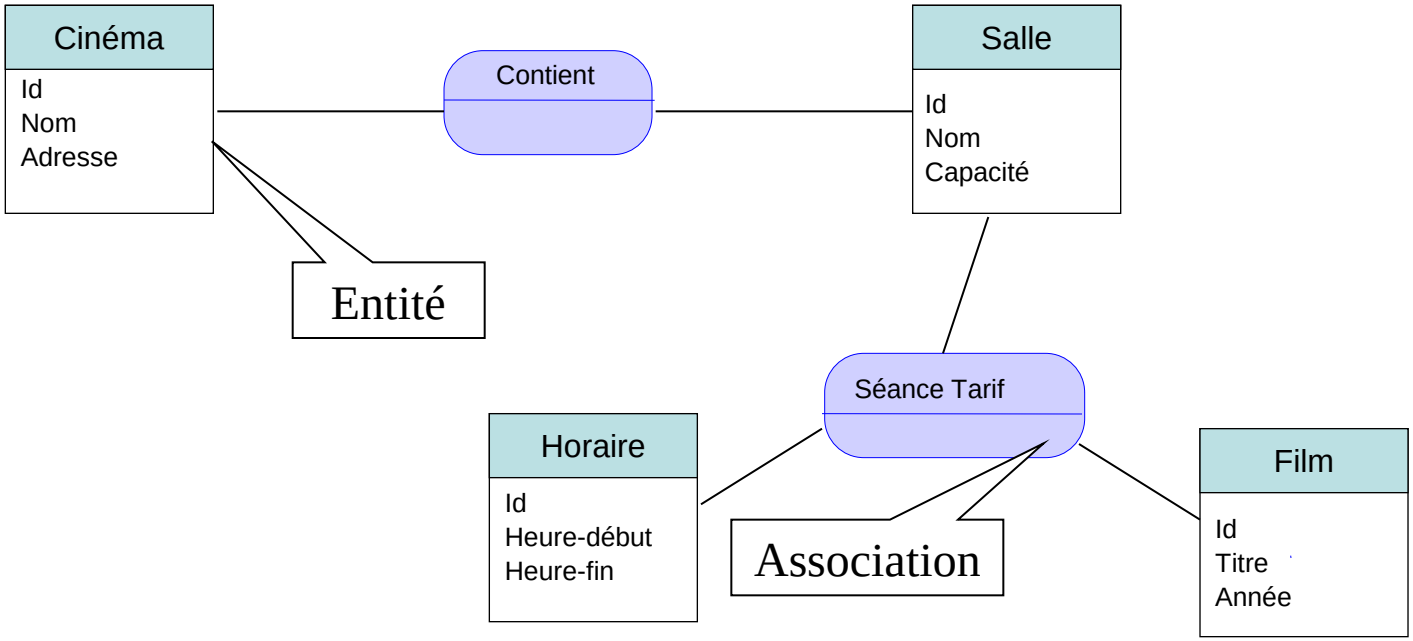
  Exemple : un livre, un étudiant, un compte, une facture, ...

- Une association est un lien entre plusieurs entités

  Exemple : une facture *contient* plusieurs produits.

MCD : Introduction

- Modèle entités-associations - Exemple:



Entités

- Entités :

- Une entité est la représentation d'un élément matériel ou immatériel ayant un rôle dans le système que nous désirons décrire

- Il existe une occurrence UNIQUE de l'entité

- Chaque entité possède:
  - Un Type ou bien une classe d'entité (similaire à un objet, alors le type de l'entité est similaire à la notion de classe)
  - Des propriétés (un ou plusieurs attributs)

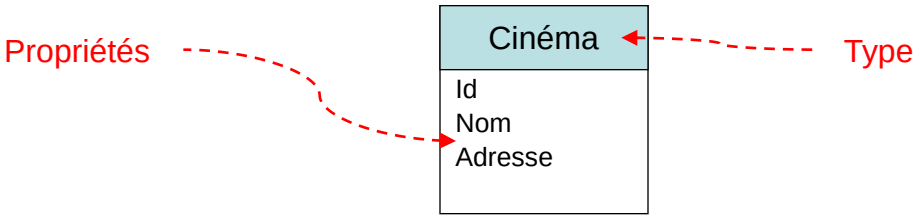
Entités

- Entités – représentation graphique :

➢ Une entité est représentée par un rectangle divisé en 2 parties :

– 1ère partie : le type (intitulé)

– 2ème partie : les propriétés (un ou plusieurs attributs)



Entités

- Entités – représentation graphique :

- Un *identifiant* est un ensemble de propriétés (une ou plusieurs) permettant de désigner une et une seule entité

- Les attributs d'une classe d'entité permettant de désigner de façon unique chaque instance de cette entité sont appelés identifiants absolus

- Le modèle conceptuel des données propose de faire précéder d'un # les identifiants (parfois de les souligner).

Entités

- Identifiant – remarques :

➢ Évitez les identifiants composés de plusieurs attributs (comme, par exemple, un identifiant formé par les attributs nom et prénom) :
  - ils dégradent les performances du SGBD,
  - mais surtout l'unicité supposée par une telle démarche finit généralement, tôt ou tard, par être démentie

➢ Évitez les identifiants susceptibles de changer au cours du temps (comme la plaque d'immatriculation d'un véhicule)

➢ Évitez les identifiants du type chaîne de caractère

Associations

- Associations :

- Une association (appelée aussi parfois relation) représente les liens sémantiques qui peuvent exister entre plusieurs entités

- Une classe (ou bien type) d'association contient toutes les associations similaires (qui relient donc des entités appartenant à des mêmes classes d'entité)

- Une classe d'association peut lier plus de deux classes d'entité. Voici les dénominations des classes d'association :
  - une classe d'association *récursive* (ou réflexive) relie la même classe d'entité
  - une classe d'association *binaire* relie deux classes d'entité
  - une classe d'association *ternaire* relie trois classes d'entité
  - une classe d'association *n-aire* relie n classes d'entité

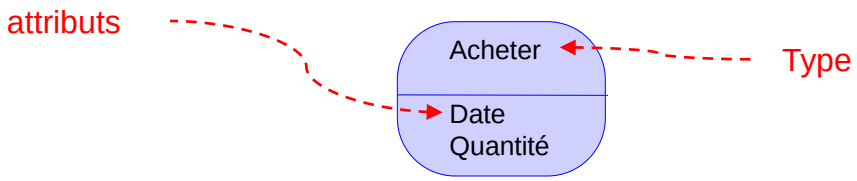
Associations

- Associations – représentation graphique :

- Une association est représentée par un cercle, ellipse ou par un rectangle à coins arrondis

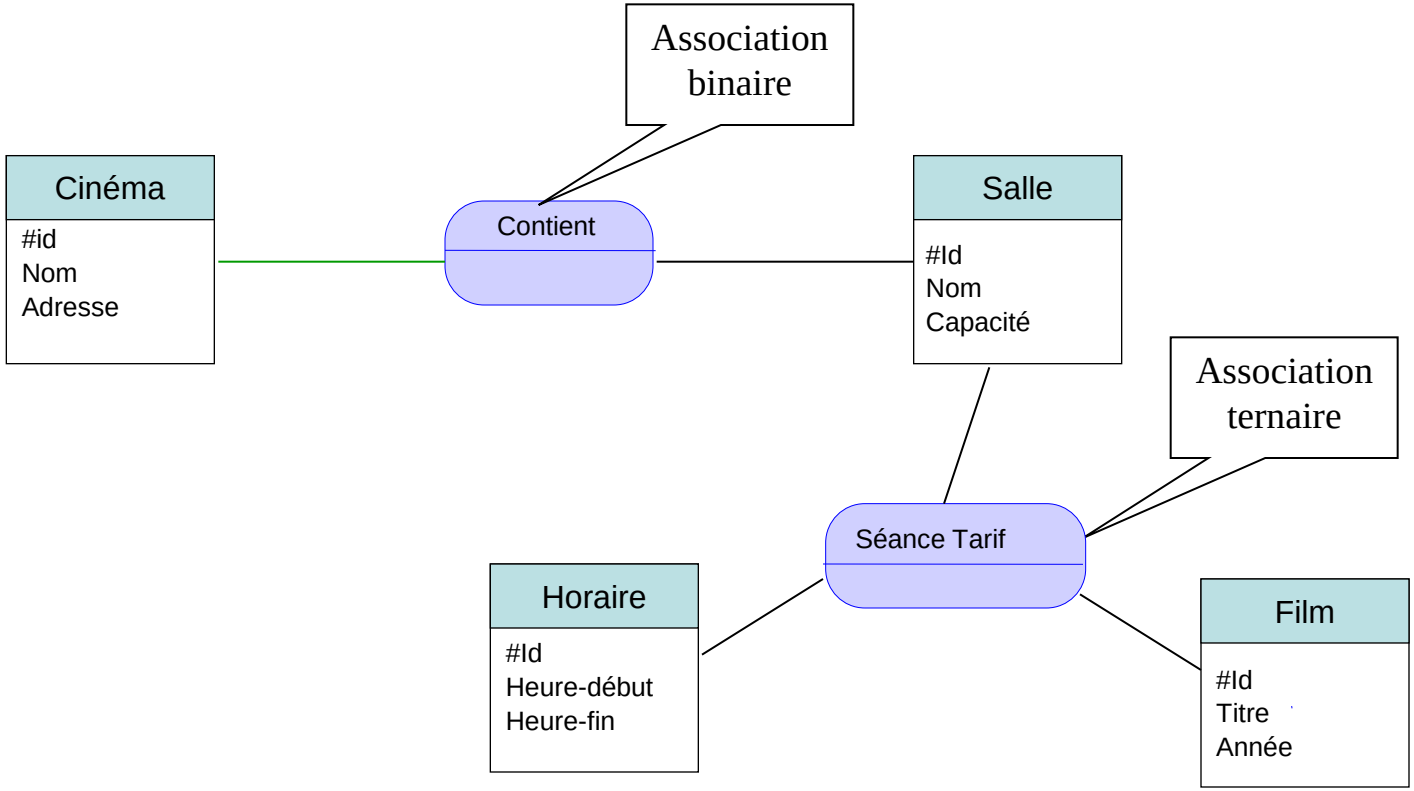
- L'intitulé décrit le type d'association qui relie les classes d'entité (généralement un verbe)

- Comme pour la classe d'entité, une classe (ou bien type) d'association peut avoir ses propres attributs



Associations

- Associations – Exemple :

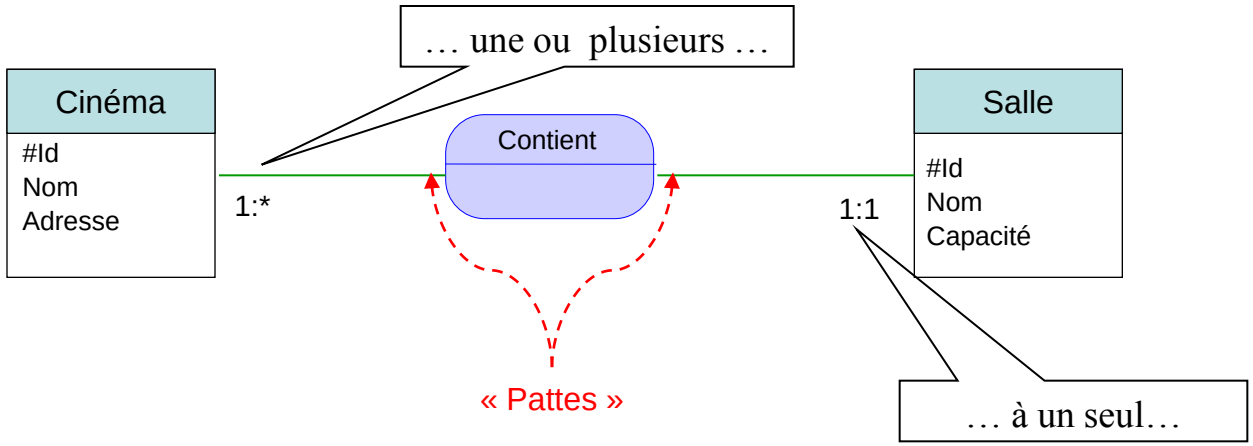


Associations

- Associations - suite:

- Remarque : *une association est composée de pattes. Chaque patte est associée à une entité et permet d'indiquer le rôle joué par celle-ci dans la relation*

- Un cinéma contient *une ou plusieurs* salles
- Une salle appartient *à un seul* cinéma



Cardinalité

- Cardinalité :

➢ La cardinalité d'un lien entre une entité et une association est le minimum et le maximum de fois qu'une entité peut être concernée par l'association:

  – la borne minimale (généralement 0 ou 1) décrit le nombre minimum de fois qu'une entité peut participer à une relation,

  – la borne maximale (généralement 1 ou n) décrit le nombre maximum de fois qu'une entité peut participer à une relation

- Exemple :

  – 1,n : signifie que chaque entité appartenant à une classe d'entité participe au moins une fois à la relation

  – 0,n : signifie que chaque entité appartenant à une classe d'entité ne participe pas forcément à la relation

Cardinalité

- Cardinalité :

➢ La représentation graphique de la cardinalité peut se faire de la manière suivante :

0:n 0..n 0,n

- Cas possibles :

0,1	Une occurrence participe au moins 0 fois et au plus 1 fois à l'association
1,1	Une occurrence participe exactement 1 fois à l'association
0,N	Une occurrence peut ne pas participer ou participer plusieurs fois
1,N	Une occurrence participe au moins 1 fois, voire plusieurs

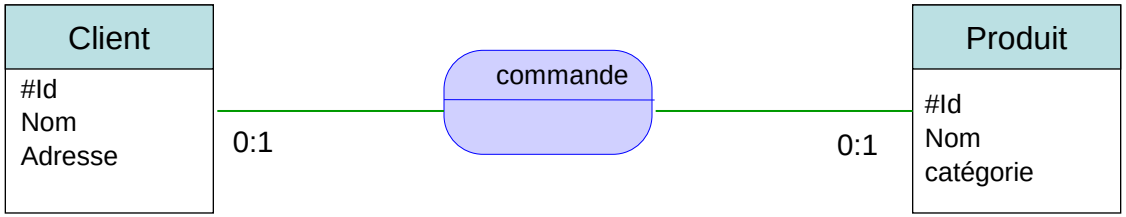
Cardinalité

- Cardinalité – exemple :

➢ Cardinalité 0,1 :

– Un client a donné commande d'un produit mais peut ne pas faire commande

– Un produit donné est commandé au maximum par un seul client mais peut ne pas être commandé



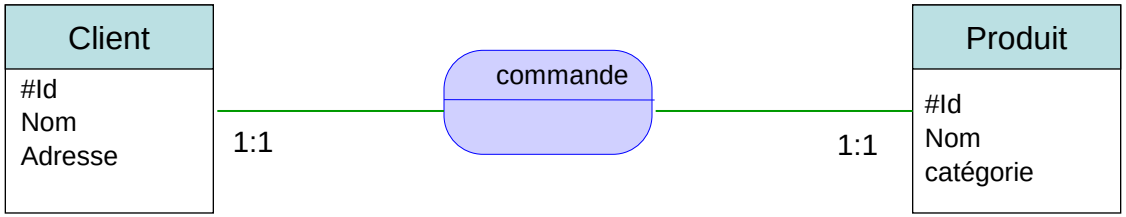
Cardinalité

- Cardinalité – exemple :

➢ Cardinalité 1,1 :

– Un client donné ne commande qu'un seul produit

– Un produit donné n'est commandé que par un seul client



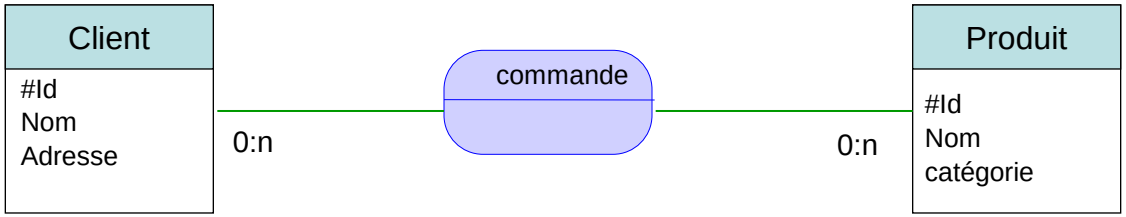
Cardinalité

- Cardinalité – exemple :

➢ Cardinalité 0,n :

– Un client a donné commande de plusieurs produits mais peut ne pas faire commande

– Un produit donné est commandé par plusieurs clients mais peut ne pas être commandé



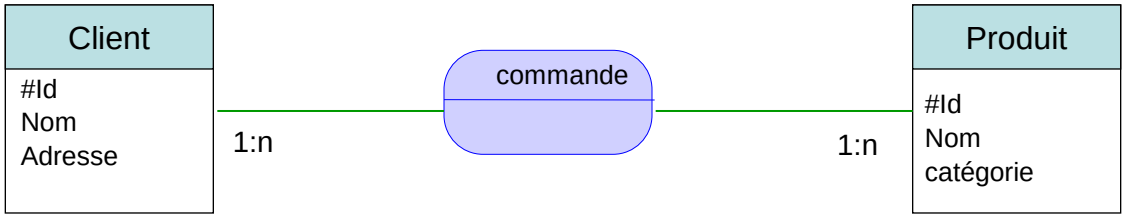
Cardinalité

- Cardinalité – exemple :

➢ Cardinalité 1,n :

– Un client a donné commande de plusieurs produits

– Un produit donné est commandé par plusieurs clients

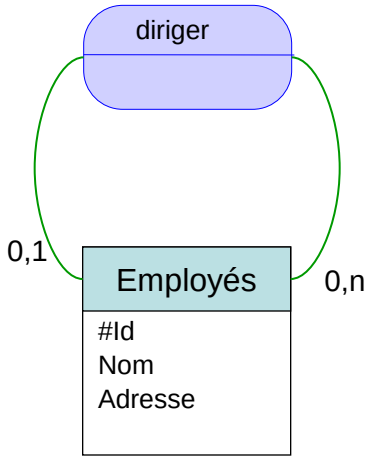


Cardinalité

- E\A – cas particuliers - exemple :

➢ Association récursive :

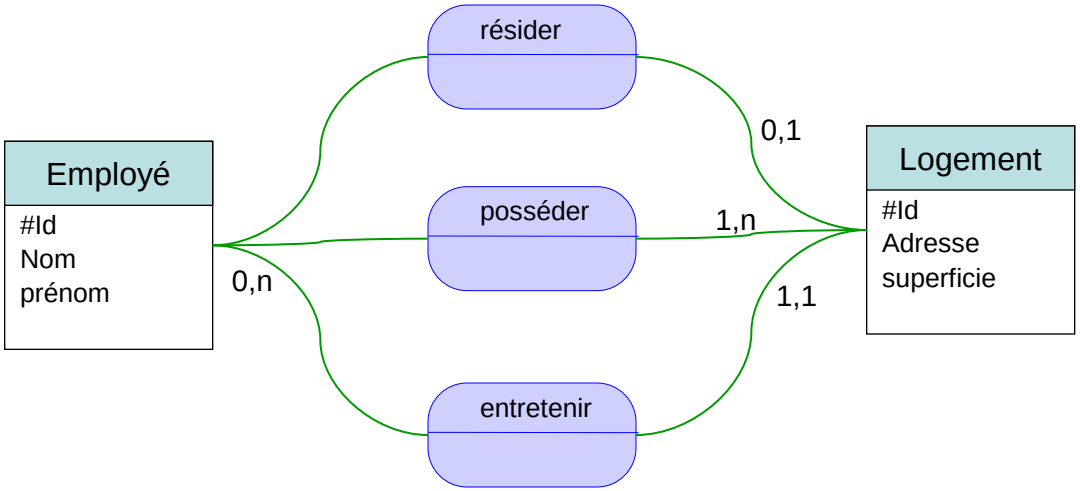
- Un employé est dirigé par un autre employé (sauf le directeur général) et un employé peut diriger plusieurs employés



Cardinalité

- E\A – cas particuliers - exemple :

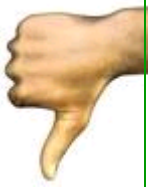
- Association multiple entre entités:
  - Un employé peut être propriétaire, locataire et/ou Chargé de l'entretien d'un logement



Résumé

- Modèle entité / association – résumé :

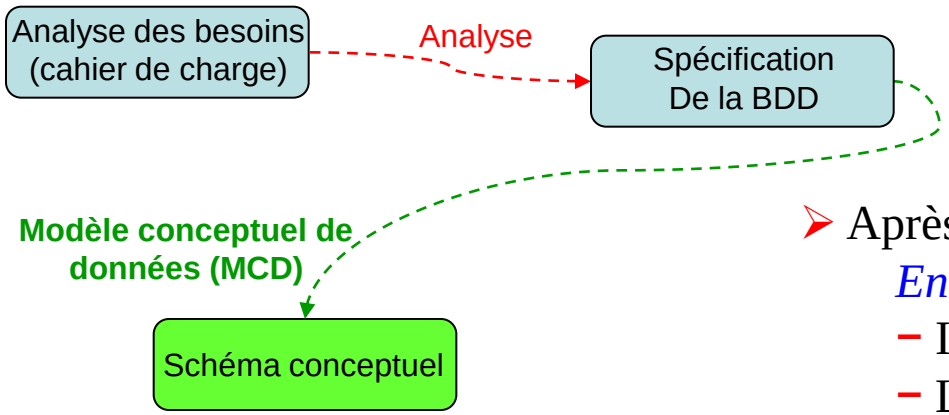
- Simple et pratique (utile pour la phase de conception)
- Indépendant du modèle logique
- 3 concepts: entité, relations et attributs
- Représentation graphique intuitive
- Modélisation pas trop complexes



- Non déterministe: pas de règle absolue pour déterminer ce qui est entité, attribut ou relation
- Pas de possibilité d'exprimer des contraintes
- Modélisation difficiles des structures complexes

Méthodologie

- Méthodologie de construction d'un MCD :



- Après l'analyse du problème, il faut :

  Entités :
  - Identifier les entités
  - Lister leurs attributs
  - Ajouter les identifiants

  Associations :
  - Établir les relations entre les entités
  - Lister les attributs des relation

  Cardinalité :
  - Calculer les cardinalités
- Résultat : *schéma conceptuel*

MCD - Exemple

- Méthodologie de construction d'un MCD - exemple :

- Exemple détaillé :

➢ Problème : on veut construire une base de données contenant les informations sur les consultations des patients ainsi que sur les médicaments prescrits par les médecins traitants

MCD - Exemple

- Méthodologie de construction d'un MCD - exemple :

➢ Les entités :

Une entité est la représentation d'un élément matériel ou immatériel ayant un rôle dans le système que nous désirons décrire

❖ Éléments matériel ?

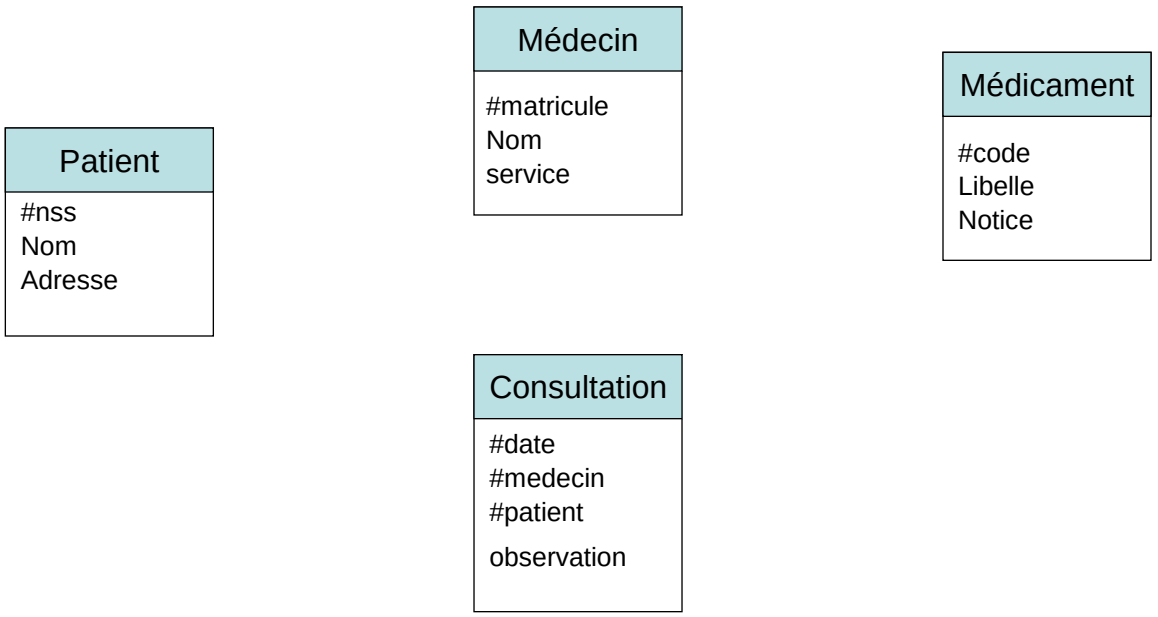
❖ Éléments immatériel ?

MCD - Exemple

- Méthodologie de construction d'un MCD - exemple :

➢ Les entités :

Patient, *Médecin*, *Médicament* et *Consultation*



MCD - Exemple

- Méthodologie de construction d'un MCD - exemple :

➢ Les relations :

Une association représente les liens sémantiques qui peuvent exister entre plusieurs entités

Association	*Patient*	*Médecin*	*Consultation*	*Médicament*
Patient	*Parent*	*Ausculter*	*Assiste*	*Prend*
Médecin	*Ausculte*	*Responsable*	*Donne*	*Prescrit*
Consultation	*Faite sur*	*Faite par*	-	*Traitement*
Médicament	*Prescrit*	*donner*	*Prescrit*	-

MCD - Exemple

- Méthodologie de construction d'un MCD - exemple :

➢ Les relations :

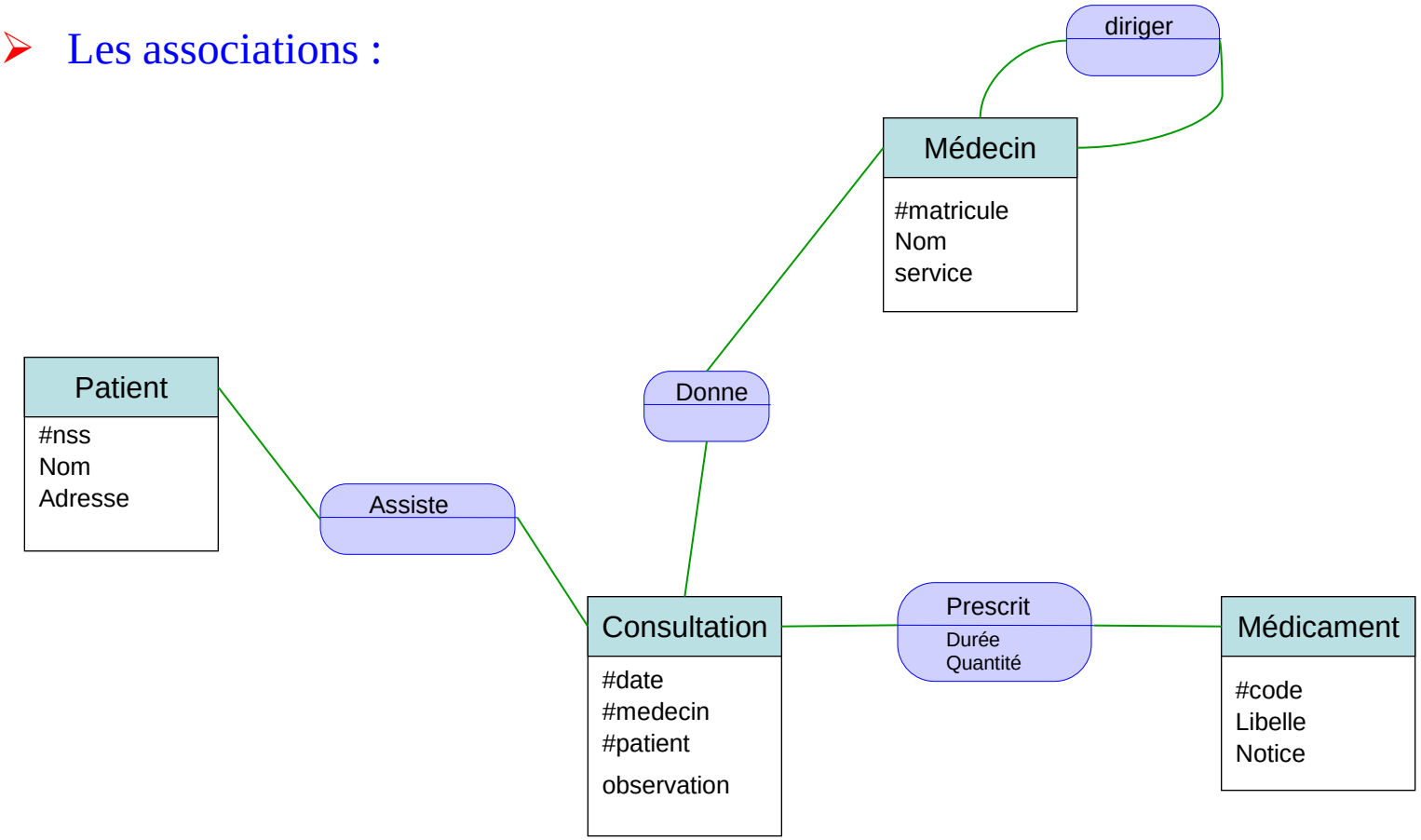
Éliminer la redondance

Association	*Patient*	*Médecin*	*Consultation*	*Médicament*
Patient	*Parent*	*Ausculter*	*Assiste*	*Prend*
Médecin	*Ausculte*	*Responsable*	*Donne*	*Prescrit*
Consultation	*Faite sur*	*Faite par*	-	*Traitement*
Médicament	*Prescrit*	*donner*	*Prescrit*	-

MCD - Exemple

- Méthodologie de construction d'un MCD - exemple :

➢ Les associations :



MCD - Exemple

- Méthodologie de construction d'un MCD - exemple :

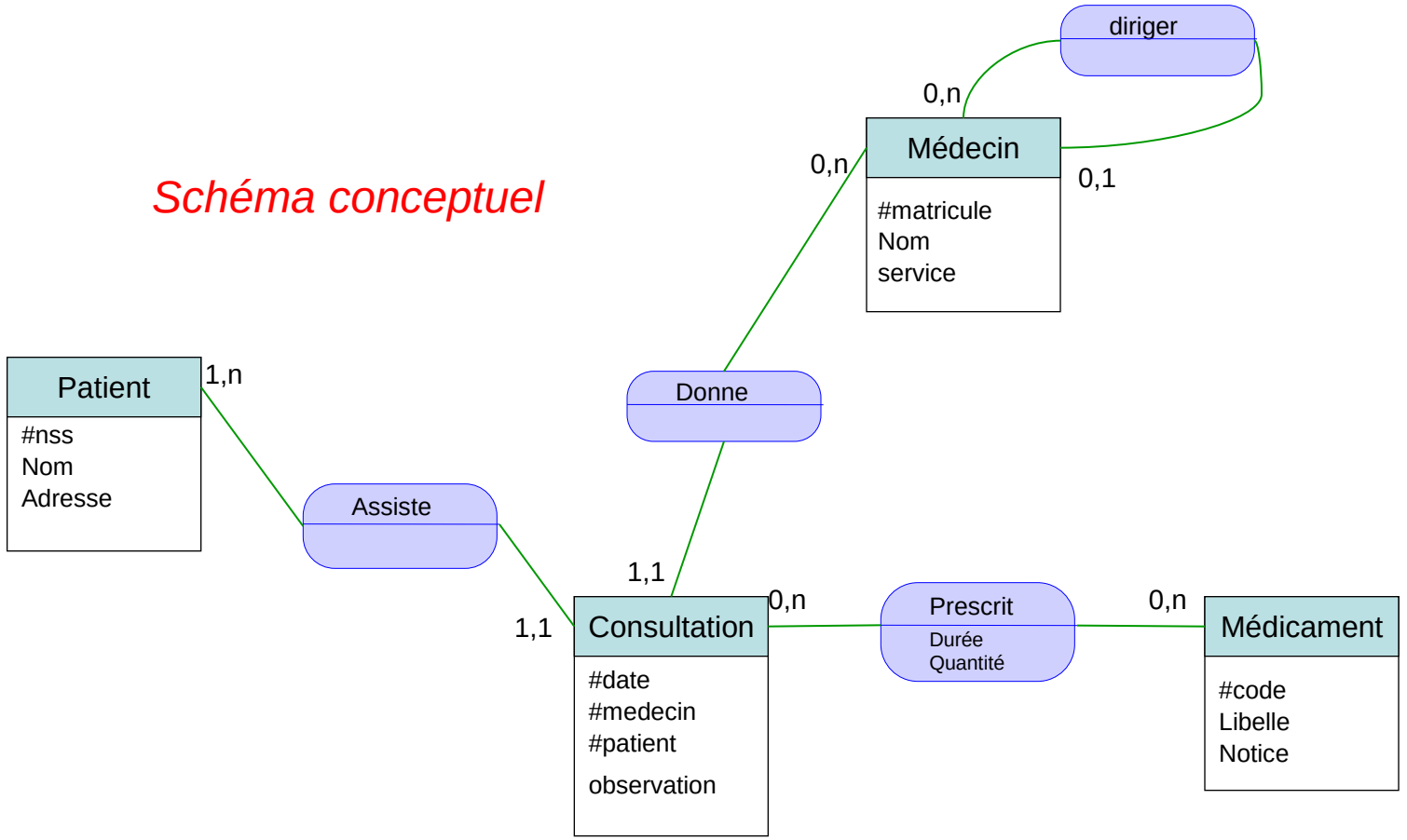
➢ Les cardinalités :

Il faut répondre à un certain nombre de questions :

- Un patient peut-il effectuer plusieurs visites ?
- Un médecin peut-il recevoir plusieurs patients dans la même consultation ?
- Peut-on prescrire plusieurs médicaments dans une même consultation ?
- Deux médecins différents peuvent-ils prescrire le même médicament ?
- ….

MCD - Exemple

- Méthodologie de construction d'un MCD - exemple :



Conseils de modélisation

- Quelques conseils pour la construction d'un MCD:

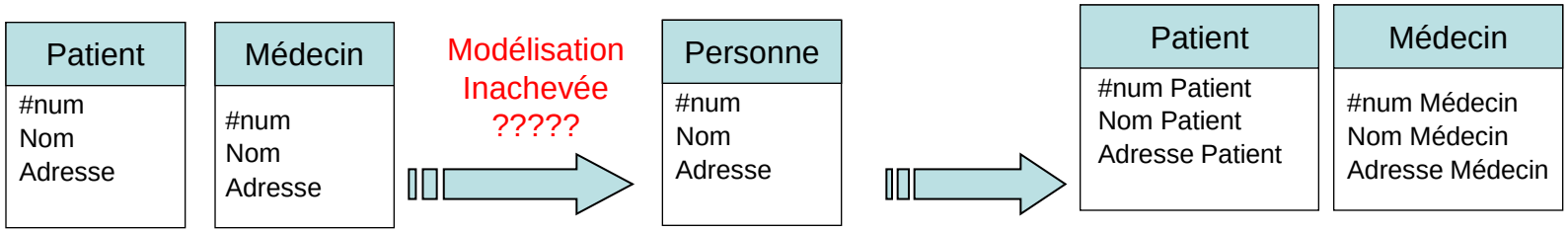
➢ Il faut prendre en compte quelques recommandations :

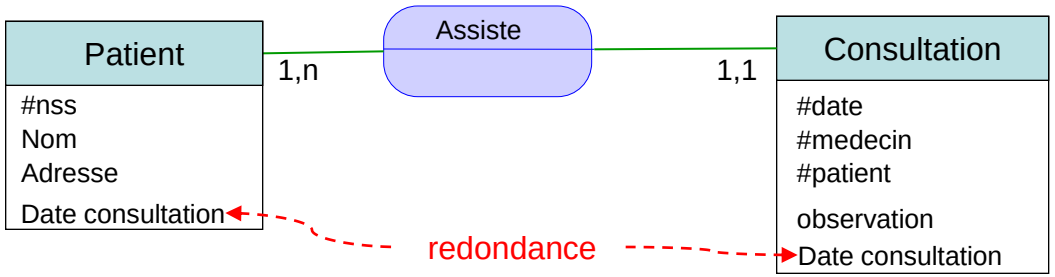
- Normalisation des noms
- Normalisation des attributs
- Faire des fusions quand c'est possible
- Faire des suppressions quand c'est possible

Conseils de modélisation

- Normalisation des noms :

  - Le nom doit être unique (entités, associations, attributs)
  - Lorsque des attributs portent le même nom, c'est parfois le signe d'une modélisation inachevée ou d'une redondance (elle entraîne un gaspillage d'espace mémoire mais aussi et surtout un grand risque d'incohérence)

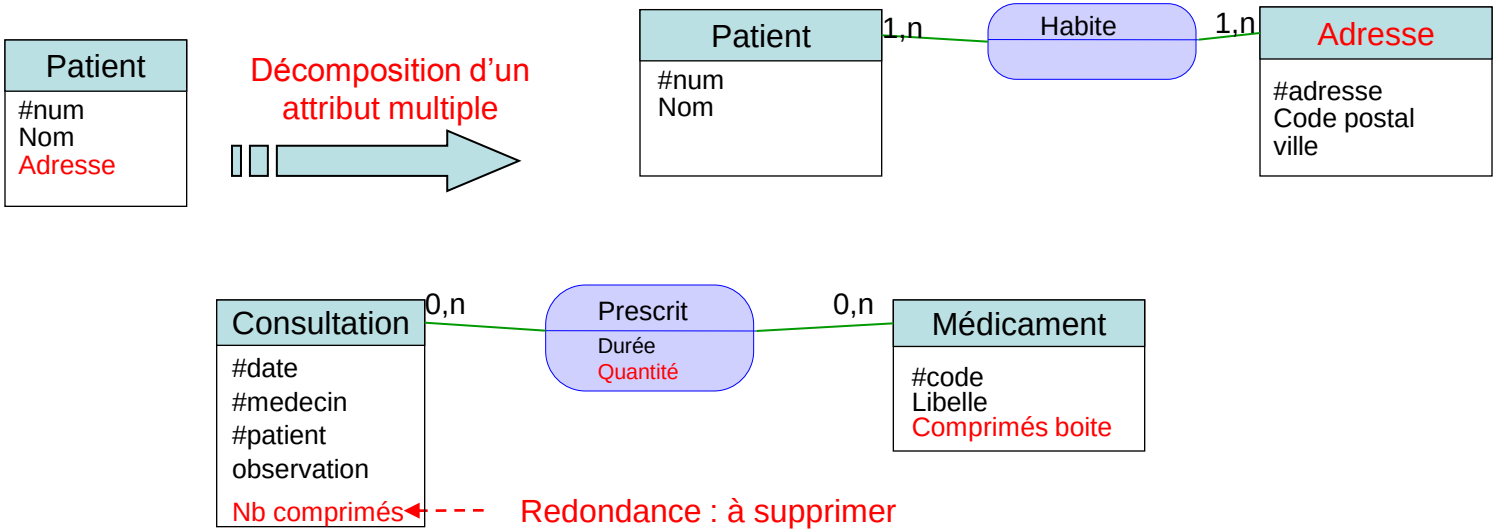




Conseils de modélisation

- Normalisation des attributs :

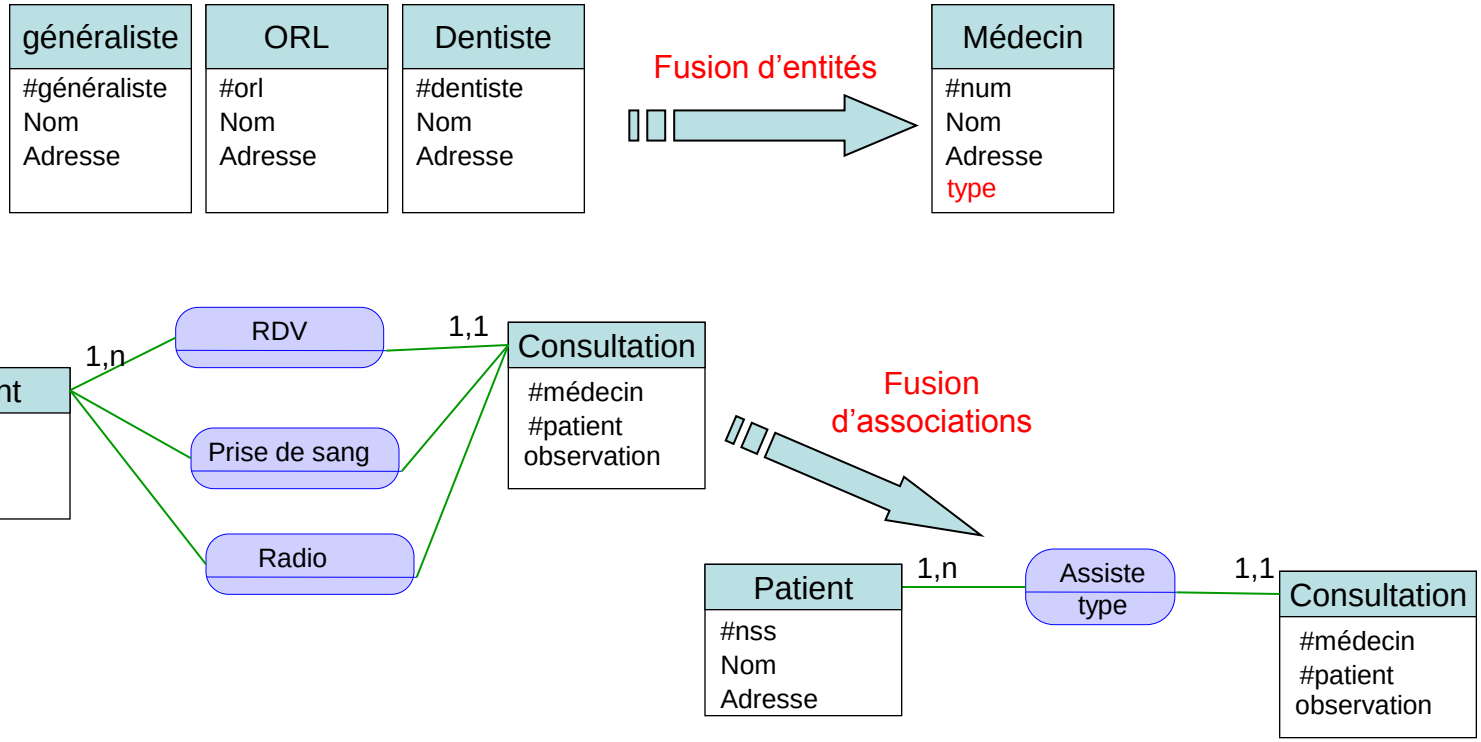
  - Chaque attribut multiple (structure) doit être transformé en une entité et une association supplémentaire est rajoutée
  - Idem pour une énumération (ex. type ={généraliste, ORL, dentiste,…})
  - Eviter la redondance d'information des attributs par calcul ou transition: un attribut dérivé d'autres attributs



Conseils de modélisation

- Fusion :

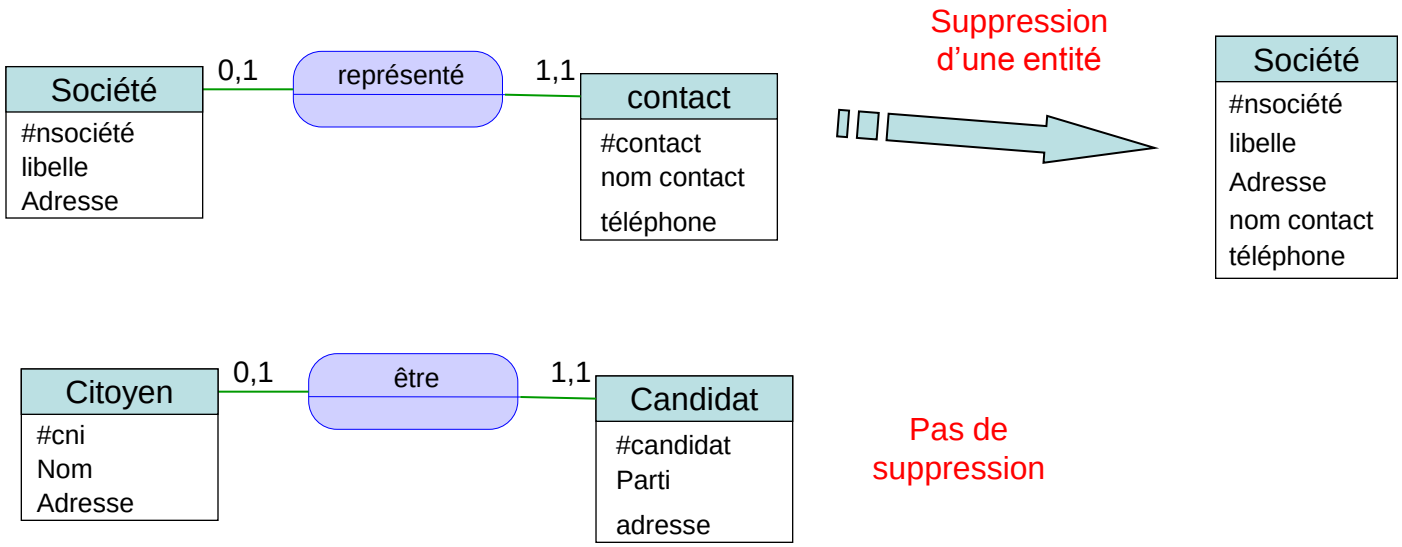
➢ Il faut factoriser les entités et les associations quand c'est possible



Conseils de modélisation

- Suppression :

➢ Il faut se poser la question de l'intérêt d'une association quand les cardinalités maximale sont toutes égales à 1



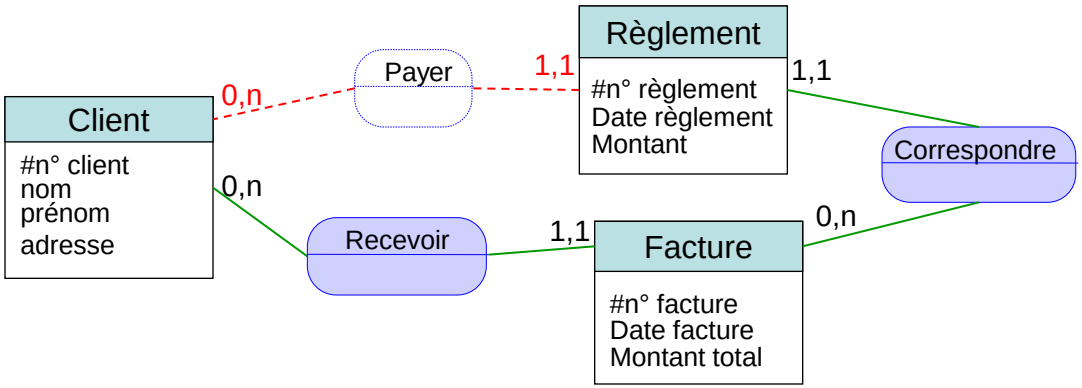
Conseils de modélisation

- Suppression :

➢ S'il existe deux chemins pour se rendre d'une entité à une autre, alors ces deux chemins doivent avoir deux significations distinctes

exemple : si un client ne peut pas régler la facture d'un autre client, alors l'association payer est inutile.

➢ Dans le cas contraire, il faut supprimer le chemin le plus court puisqu'il est déductible des autres chemins



Normalisation

- Normalisation : il existe des règles de normalisation, appelés *forme normale*

➢ Les formes normales permettent d'éviter la redondance et les sources d'anomalies

➢ Il existe plusieurs formes normales :

- ✓ 1FN : 1ère forme normale
- ✓ 2FN : 2ème forme normale
- ✓ 3FN : 3ème forme normale
- ✓ FNBC : Forme normale de Boyce-Codd
- ✓ 4FN : 4ème forme normale
- ✓ 5FN : 5ème forme normale
- ✓ …

Plus le niveau de normalisation est élevé, plus le modèle est exempte de redondances (modélisation rigoureuse)

Normalisation

- Normalisation : pourquoi???

- Exemple : Relation Commande_Produit

N° produit	Quantité	N° Fournisseur	Nom fournisseur
1023	10	123	Elvis
2001	150	211	Bernard
1256	12	160	Dupont
1236	50	123	Elvis
….	….	….	

- Problèmes (anomalies):

➢ Une maj de l'adresse d'un fournisseur → modification de toutes les enregistrements concernés

➢ L'insertion d'un nouveau fournisseur → obligation de faire une commande

➢ La suppression d'un produit → suppression d'un fournisseur

Normalisation

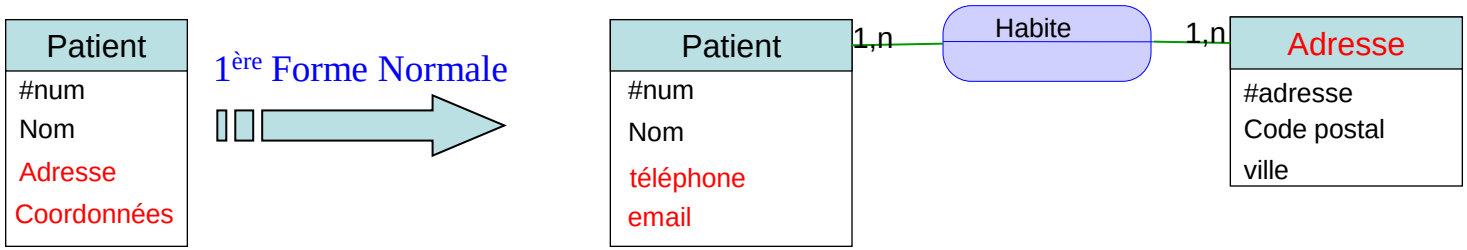
- Normalisation (*forme normales):*

➢ 1ère forme normale (1FN) : Une entité ou une association est en première forme normale si tous ses attributs sont *élémentaires*, c'est-à-dire non décomposables

➢ Un élément composite doit être décomposés en :

✓ Attributs élémentaires (ex. coordonnées)

✓ Ajoutant une nouvelle entité (ex. adresse)

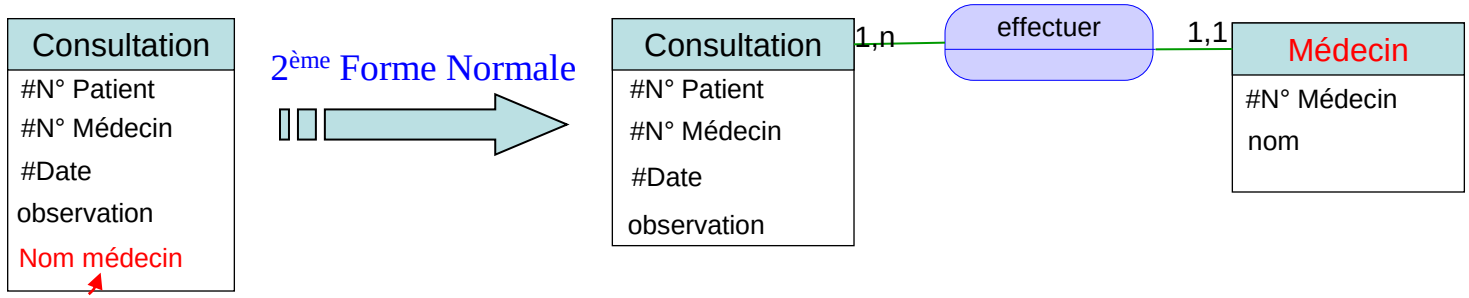


Normalisation

- Normalisation (*forme normales):*

➢ 2ème forme normale (2FN) : une entité ou une association est en deuxième forme normale si, et seulement si, elle est en 1ère forme normale et si tout attribut n'appartenant pas à l'identifiant ne dépend pas d'une partie de l'identifiant

➢ Autrement dit, l'identifiant peut être composé de plusieurs attributs mais les autres attributs de l'entité doivent être dépendant de l'identifiant en entier (et non pas une partie de cet identifiant)



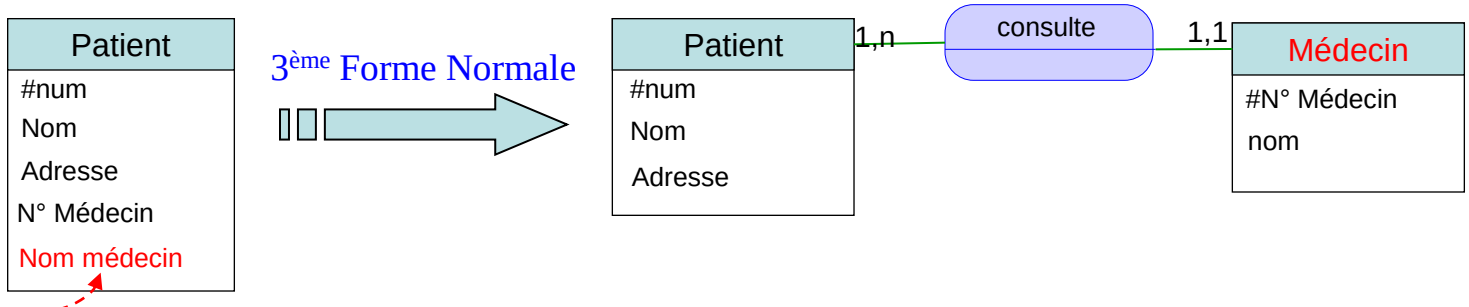
L'attribut « Nom médecin » dépend de l'attribut « N° Médecin » (une partie de l'identifiant) : *N° Médecin → Nom médecin*

Normalisation

- Normalisation (*forme normales):*

➢ 3ème forme normale (3FN) : une entité ou une association est en troisième forme normale si, et seulement si, elle est en 2ème forme normale et si tout attribut n'appartenant pas à l'identifiant ne dépend pas d'un autre attribut

➢ Garantir que seulement l'identifiant détermine tous les attributs d'une relation



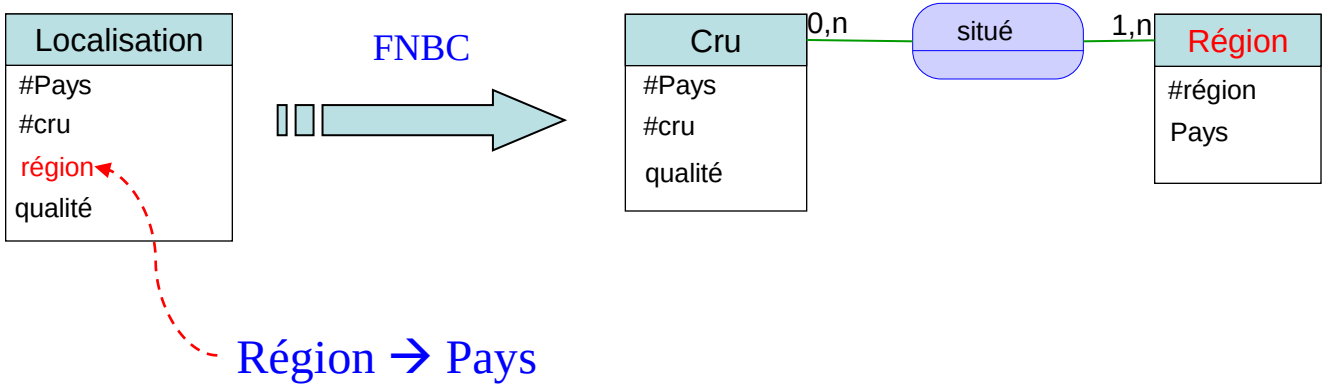
L'attribut « Nom médecin » dépend de l'attribut « N° Médecin »:
N° Médecin → Nom médecin

Normalisation

- Normalisation (*forme normales):*

➢ Forme normale de Boyce-Codd (FNBC) : les attributs d'un identifiant composé ne doivent pas dépendre d'un autre attribut de l'entité

➢ Une entité ou une association est en forme normale de Boyce-Codd si, et seulement si, elle est en troisième forme normale et si aucun attribut faisant partie de l'identifiant ne dépend pas d'un autre attribut ne faisant pas partie de l'identifiant



Normalisation -exemple

- Normalisation – formes normales :

- Exemple détaillé :

➢ Problème : on veut vérifier la normalisation de l'entité étudiant suivante

✓ Tarif : tarif CROUS {T1, T2, T3} selon l'âge de l'étudiant

✓ Enseignant : enseignant responsable de la matière

✓ Moyenne : moyenne dans une matière

Etudiant
#codeEtudiant
nom
âge
ville
tarif
#matière
enseignant
moyenne

Etudiant(codeEtudiant, nom, age, ville, tarif, matière, enseignant, moyenne)

Normalisation -exemple

- Normalisation – formes normales :

- Exemple détaillé :

➢ Les dépendances :

codeEtudiant → *nom, âge, ville*

matière → *enseignant*

codeEtudiant, matière → *moyenne*

âge → *tarif*

Etudiant
#codeEtudiant
nom
âge
Ville
tarif
#matière
enseignant
moyenne

Question : est-ce que l'entité est en 1ère forme normale ????

tous ses attributs sont *élémentaires* ➔ *oui*

Normalisation -exemple

- Normalisation – formes normales :

- Exemple détaillé :

➢ Les dépendances : ***codeEtudiant*** → *nom*, *âge*, *ville*

matière → *enseignant*

codeEtudiant, *matière* → *moyenne*

âge → *tarif*

Etudiant
#codeEtudiant
nom
âge
ville
tarif
#matière
enseignant
moyenne

Question : est-ce que l'entité est en 2ème forme normale ????

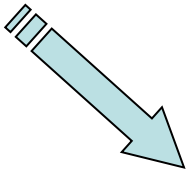
tout attribut n'appartenant pas à l'identifiant ne dépend pas d'une partie de l'identifiant ➔ *non*

Normalisation -exemple

- Normalisation – formes normales :

- Exemple détaillé :

➢ Décomposition :



Etudiant
#codeEtudiant
nom
âge
ville
tarif

Matière
#matière
Enseignant

Note
#codeEtudiant
#matière
moyenne

❑ Les dépendances :

codeEtudiant → nom, âge, ville

âge → tarif

Matière → enseignant

codeEtudiant, matière → moyenne

Question : est-ce que les entités sont en 3ème forme normale ????

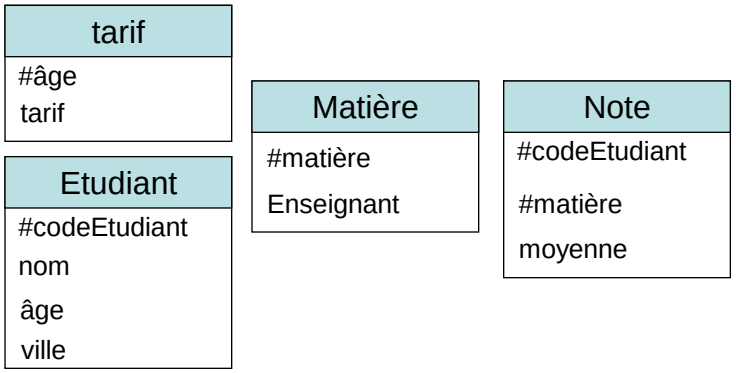
tout attribut n'appartenant pas à l'identifiant ne dépend pas d'un autre attribut ➔ *non*

Normalisation -exemple

- Normalisation – formes normales :

- Exemple détaillé :

➢ Décomposition :



Les entités sont en 3ème forme normale

Question : est-ce que les entités sont en forme normale Boyce Codd ????

Normalisation - conclusion

- Normalisation – conclusion :

➢ Il existe d'autres formes normales comme la forme normale domaine-clé (FNDC), la forme normale de restriction-union ou la sixième forme normale (6NF)

➢ Bien que l'objectif de la normalisation soit d'amener le concepteur à obtenir des relations en forme normale finale, cet objectif ne doit pas être interprété comme une loi

➢ D'autres facteurs sont à considérer dans le processus de conception d'une base de données et l'expérience et l'intuition jouent un rôle important

Partie 3 : Modèle Logique des Données (MLD)

- Introduction
- Les objets relationnels
- Les structures de données
- Règles de passage d'un MCD à un MLD
- Intégrités des données

MLD : Introduction

- Modèle logique des données (MLD):

- Le modèle logique des données consiste à décrire la structure de données utilisée sans faire référence à un langage de programmation. Il s'agit donc de préciser le type de données utilisées lors des traitements

- Ainsi, le modèle logique est dépendant du type de base de données utilisé

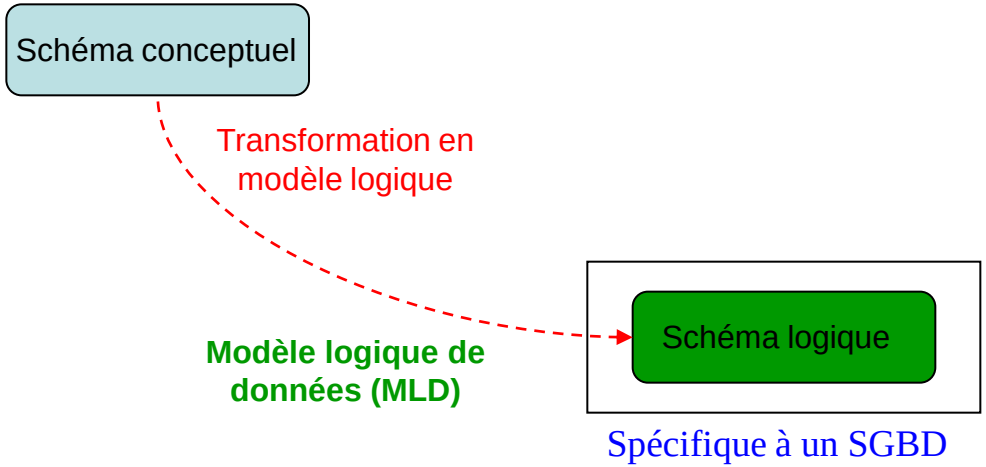
- Il s'obtient en transformant le modèle conceptuel de données

- Le passage du modèle conceptuel au modèle logique, se fait en suivant un certain nombre de règles de *traduction*

MLD : Introduction

- Modèle logique des données (MLD):

- Le MLD adapte le MCD au type de SGBD utilisé
- Pour les SGBD relationnels, le modèle conceptuel est traduit sous forme de relations
- Pour les SGBD objets, le modèle conceptuel est traduit sous forme de classes
- Dans ce cours, nous allons traité le modèle logique de données *relationnel*



Modèle relationnel

- Modèle relationnel : introduit par E.F. Codd en 1970 (IBM)

➢ Objectifs:
  - ✓ Permettre un haut degré d'indépendance entre les programmes d'applications et la représentation interne des données,
  - ✓ Fournir une base solide pour traiter les problèmes de cohérence et redondance de données
  - ✓ Permettre le développement de langages de manipulation de données ensemblistes basés sur des théories solides (algèbre relationnelle)

➢ Entre 1975 et 1980 IBM se lance dans l'implantation d'un SGBD basé sur le modèle relationnel : *SYSTEM/R*

➢ Entre temps, J.L Ellison a créé une start-up, en se basant sur les articles de Codd, qui va vendre des SGBD basées sur le modèle relationnel : *Oracle*

Objets relationnels

- Le modèle relationnel est basé sur la notion d'ensemble (set).

- Ce modèle comporte trois notions de bases:

➢ Le domaine : ensemble de valeurs cratérisées par un nom

➢ La relation : sous ensemble du produit cartésien d'une liste de domaines caractérisée par un nom

➢ L'attribut : colonne d'une relation caractérisée par un nom

Structures de données de base

- Le domaine (*domain*) : ensemble de valeurs caractérisé par un nom

- Domaine définis en intention:
  Exemple : Entier, réel, caractères, monnaie, temps

- Domaine défini en extension:
  Exemple :

  Niveau = {1A, 2A}

  Matière = {Maths, Elec, Info}

- Produit cartésien de domaines D1 x D2 x …

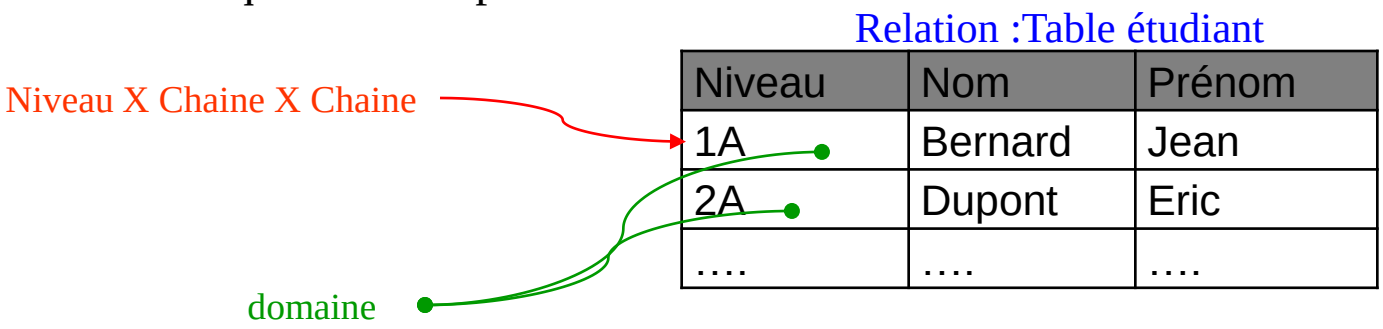
  Exemple: Niveau x Matière

  1A Maths
  1A Elec
  1A Info
  2A Maths
  ….

Structures de données de base

- La relation (*relation*) : sous-ensemble d'un produit cartésien composée de vecteurs

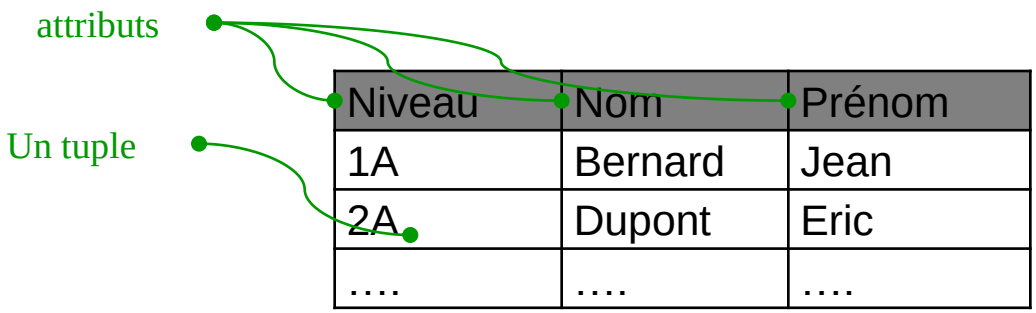
- Correspond à ce que nous avons appelé jusqu'ici une table
- Représentée sous la forme d'une *table*
- Chaque ligne correspond à un *vecteur* du produit cartésien
- Chaque *colonne* correspond à *un domaine*
- Une colonne *prend valeur* dans un *domaine*
- Chaque colonne possède un *nom*



Structures de données de base

- L'attribut (*attribute*) : correspond à une colonne d'une table
  - Permet la distinction entre domaine et colonne
  - Le nom de l'attribut doit être porteur de sens (significatif)

- Le tuple : correspond à une ligne d'une table
  - Ligne d'une relation correspondant à un *enregistrement*



Structures de données de base

- Remarques :

  - Le nombre d'attributs est appelé : *degré*
  - Le nombre de tuples est appelé : *cardinalité*

- Exemple :

Degré = 3

Cardinalité =2

Niveau	Nom	Prénom
1A	Bernard	Jean
2A	Dupont	Eric

Structures de données de base

- La clé (*key*) : est un ensemble d'attribut dont la connaissance des valeurs permet d'identifier un *Tuple* unique de la relation considérée

- La clé permet de respecter la règle de l'unicité

- Il ne doit donc pas y avoir d'éléments en *double*

  ➔ *seule valeur de clé*

étudiant

Clé : unique

numéro	Nom	Prénom
20081001	Bernard	Jean
20081002	Dupont	Eric
….	….	….

Structures de données de base

- Il existe plusieurs types de clés :

➢ La clé candidate : sous-ensemble minimal d'attributs qui permet d'identifier chacun des enregistrements de la relation

➢ La clé primaire (*primary key*): une clé candidate sélectionnée pour être la clé principale d'une relation

➢ La clé alternative (*primary key*): une clé candidate non sélectionnée pour être la clé principale

➢ La clé étrangère (externe, foreign key) : est un ensemble d'attributs dans une relation qui correspond exactement à la clé primaire d'une autre relation

- Remarque : tout attribut constituant une clé primaire ne peut pas avoir une valeur nulle

Structures de données de base

- Exemple :

  - clé candidate : *(nom, prénom) , (num)*
  - clé primaire : *num*
  - clé alternative : *(nom, prénom)*

  - clé candidate : *(num,idcours)*
  - clé primaire : *(num,idcours)*
  - clé étrangère : *num (clé dans la table étudiant), idcours (table cours)*

étudiant

Num	Nom	Prénom	adresse	
101	Bernard	Jean	Avon	
102	Dupont	Eric	Paris	
….	….	….		

Notes

Num	idCours	Note		
101	10	15		
101	12	10		
….	….	….		

Structures de données de base

- Schéma de relation : descriptif d'une relation

  - Nom de la relation suivi de la liste des attributs et de la définition de leur domaine

$$R\ (A_1:D_1,\ A_2:D_2,\ldots..A_i:D_i,\ \ldots..A_n:D_n)$$

  - Un schéma représente une *intention*, c.a.d propriétés *communes et invariantes* au cours du temps

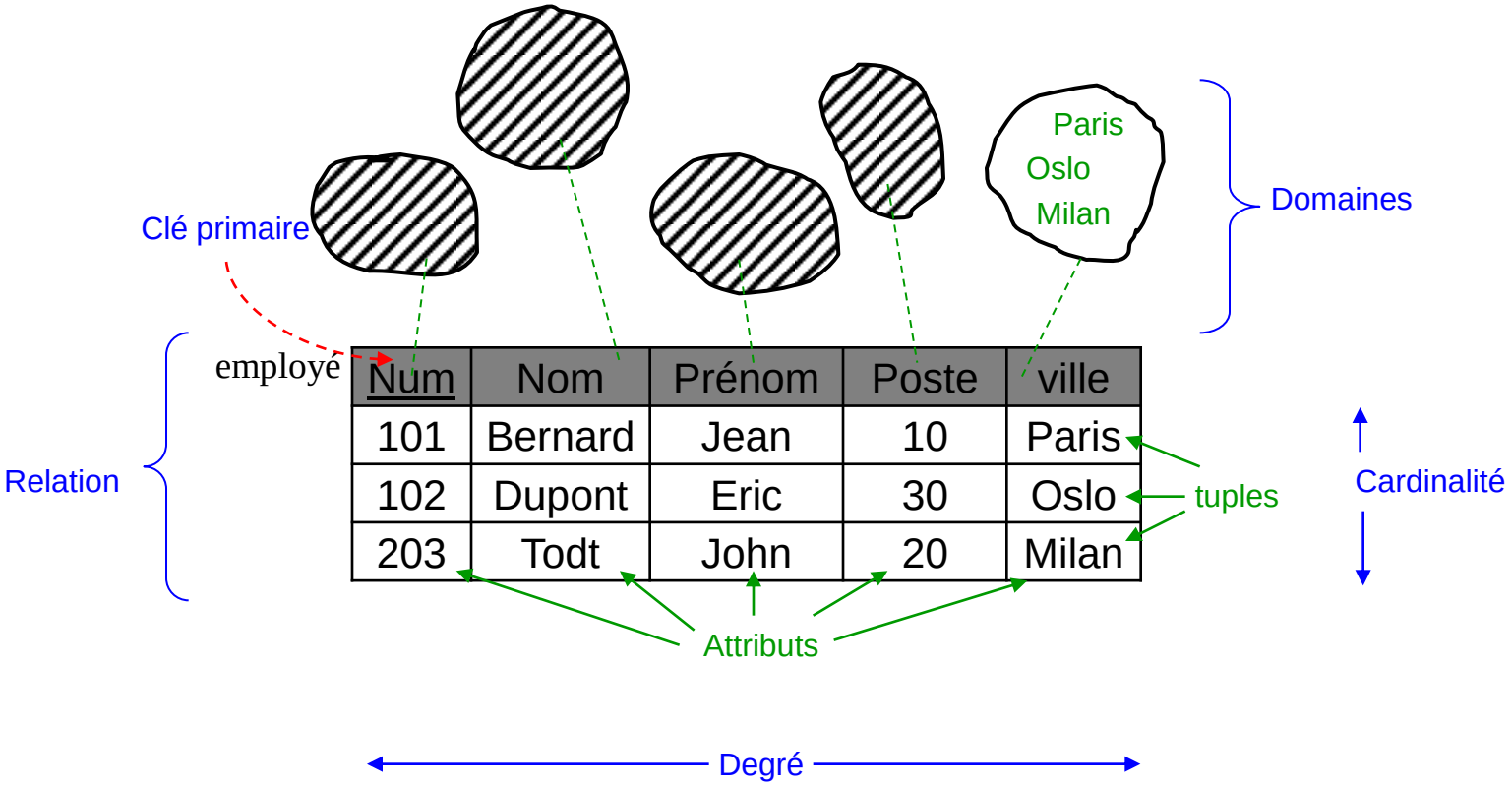
Exemple:

Etudiant (Num:INTEGER, Nom:VARCHAR, Prenom:VARCHAR, adresse:VARCHAR)

- Remarque : la clé primaire est soulignée par convention

Structures de données de base

- Exemple : la relation employé



Structures de données de base

- Terminologie : les objets relationnels

Terme relationnel formel	Equivalent informel
Relation	Table
Tuple	Ligne / enregistrement
Cardinalité	Nombre de lignes
Attribut	Colonne / champ
Degré	Nombre de colonnes
Clé primaire	Identificateur unique
Domaine	Réservoir de valeurs légales

Passage : MCD ➔MLD

- Règles de passage d'un MCD à un MLD :

➢ Le passage d'un modèle MCD en MLDR se fait en appliquant des règles :

- ✓ 1ère règle : traduction des entités
- ✓ 2ème règle : traduction des associations binaires forme 1 (x,1 ↔ x,1)
- ✓ 3ème règle : traduction des associations binaires forme 2 (x,1 ↔ x,n)
- ✓ 4ème règle : traduction des associations binaires forme 3 (x,n ↔ x,n)
- ✓ 5ème règle : traduction des associations n-aires

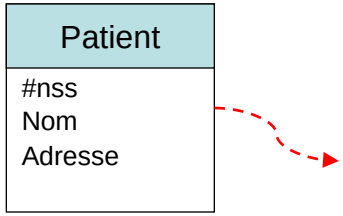
Passage : MCD ➔MLD

- Règles de passage d'un MCD à un MLD :

➢ 1ère règle : traduction des entités

- Toute entité devient une table dans laquelle les attributs deviennent des colonnes
- L'identifiant de l'entité constitue la clé primaire de la table

- Exemple :



- L'entité Patient devient une relation (table)

- Le schéma de relation :

Patient (nss, nom, adresse)

nss	nom	adresse
1807514511	Dupond	Avon
2909112000	Smith	Paris
2873914536	Pierre	Lyon

Passage : MCD ➔MLD

- Règles de passage d'un MCD à un MLD :

- Le passage du modèle conceptuel au modèle logique au niveau des classes de relation se fait selon les cardinalités des classes d'entité participant à la relation

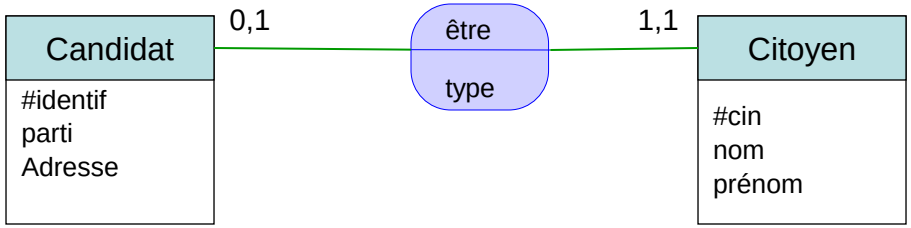
➢ 2ème règle : traduction des associations binaires (x,1 ↔ x,1)

- l'identifiant d'une entité est utilisé comme clé étrangère de l'autre relation ou inversement
- Les attributs de l'association sont alors repartis vers l'une ou l'autre des deux tables

Passage : MCD ➔MLD

- Règles de passage d'un MCD à un MLD : (2ème règle)

- Exemple :



- Les entités Candidat et Citoyen deviennent des relations

Le schéma de relation (solution 1) :
Candidat (identif, cin, parti, adresse, type)
Citoyen (cin, nom, prénom)

Le schéma de relation (solution 2):
Candidat (identif, parti, adresse, type)
Citoyen (cin, identif, nom, prénom)

Passage : MCD ➔MLD

- Règles de passage d'un MCD à un MLD :

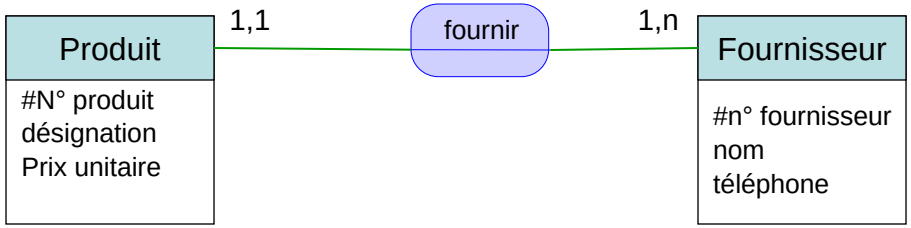
➢ 3ème règle : traduction des associations binaires (x,1 ↔ x,n)

- La Clé Primaire de la table à la cardinalité (x,n) devient une Clé Etrangère dans la table à la cardinalité (x,1)
- Les attributs de l'association sont alors repartis vers l'une ou l'autre des deux tables

Passage : MCD ➔MLD

- Règles de passage d'un MCD à un MLD : (3ème règle)

- Exemple :



- Les entités Produit et Fournisseur deviennent des relations

Le schéma de relation :

Produit (n° produit, désignation, prix unitaire, n° fournisseur)

Fournisseur (n° fournisseur, nom, téléphone)

Passage : MCD ➔MLD

- Règles de passage d'un MCD à un MLD :

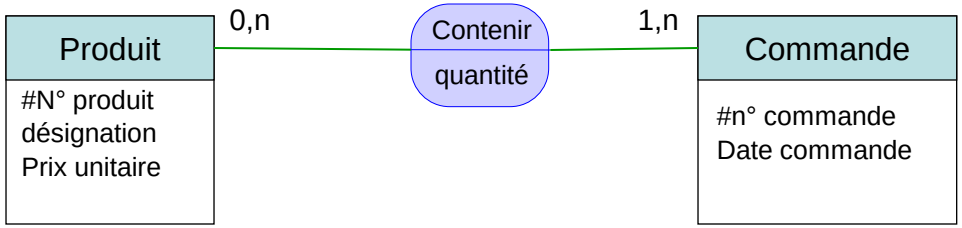
➢ 4ème règle : traduction des associations binaires (x,n ↔ x,n)

– L'association entre les deux entités est traduite par une nouvelle table dont la clé primaire est composée de deux clés étrangères vers les clés primaires des 2 tables en association
– Les attributs de l'association deviennent des attributs de cette nouvelle table

Passage : MCD ➔MLD

- Règles de passage d'un MCD à un MLD : (4ème règle)

- Exemple :



- Les entités Produit et Commande deviennent des relations

Le schéma de relation :

Produit (n° produit, désignation, prix unitaire)

Commande (n° commande, date commande)

Contenir (n° produit, n° commande, quantité)

Passage : MCD ➔MLD

- Règles de passage d'un MCD à un MLD :

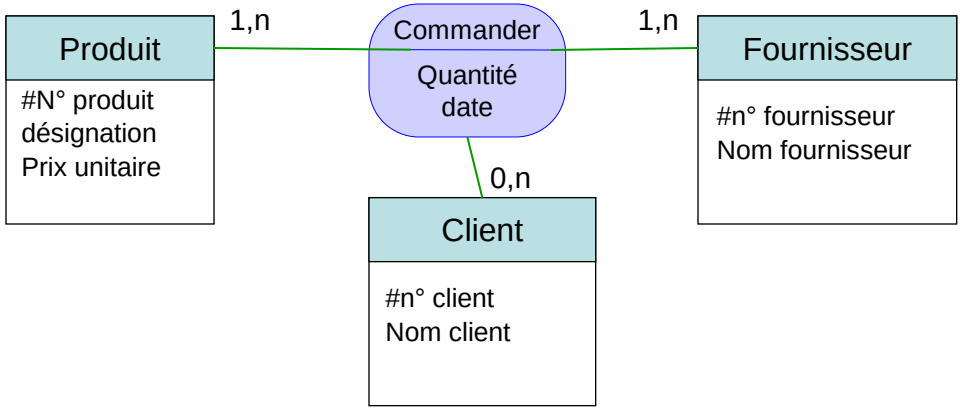
➢ 5ème règle : traduction des associations n-aires

– Une association entre trois entités ou plus est traduite par une table supplémentaire dont la clé primaire est composée d'autant de clés étrangères que d'entités
– Les attributs de l'association deviennent des attributs de cette nouvelle table

Passage : MCD ➔MLD

- Règles de passage d'un MCD à un MLD : (5ème règle)

- Exemple :



- Les entités Produit, Client et Fournisseur deviennent des relations

Le schéma de relation :

Produit (n° produit, désignation, prix unitaire)

Client(n° client, nom client)

Fournisseur(n° fournisseur, nom fournisseur)

Commander (n° produit, n° client, n° fournisseur, date commande, quantité)

Intégrités des données

- Les données contenues dans une base de données doivent respecter des règles d'intégrité

- Il est possible de distinguer les :
  - Règles structurelles qui sont inséparables au modèle de données,
  - Règles de comportement qui sont propres au schéma particulier d'une application

- Le modèle relationnel impose
  - Une règle structurelle qui est : l'unicité des clés
  - Il est commode d'ajouter deux types de règles d'intégrité en plus :
    - ✓ Contraintes de références
    - ✓ Contraintes d'entité

Intégrités des données

Contraintes de références :

- Le modèle relationnel est souvent utilisé afin de représenter des entités du monde réel qui sont des objets ayant une existence propre, et des associations entre ces objets

  - *BUVEURS(NB:ENTIER,NOM:CHAR,PRENOM:CHAR,TYPE:CHAR)*
  - *VINS (NV:ENTIER,CRU:CHAR,MILL:ENTIER,DEGRE:REEL)*
  - *ABUS (NB:ENTIER, NV:ENTIER, DATE:CHAR,QUANTITE:REEL)*

- Définition : c'est une contrainte d'intégrité portant sur une relation R1, consistant à imposer que la valeur d'un groupe d'attributs apparaisse comme valeur de clé dans une autre relation R2

Intégrités des données

- Contraintes de références :

➢ Une telle contrainte d'intégrité s'applique en général aux associations : *une association ne peut exister que si les entités participant à l'association existent déjà*

➢ La représentation de contraintes de référence peut s'effectuer par la définition de *clés étrangères* dans une relation

- Exemple : La relation ABUS

ABUS (NB:ENTIER, NV:ENTIER, DATE:CHAR,QUANTITE:REEL)

- une clé primaire : *(NB,NV,DATE)*
- deux clés étrangères *NB* et *NV*

Intégrités des données

- Contraintes d'entité :

- Un ou plusieurs attributs peuvent être inconnus lors de la création d'un Tuple, dans ce cas nous sommes amenés à introduire dans la relation une valeur conventionnelle appelée valeur nulle*(NULL value)*

- Pour respecter la contrainte d'intégrité d'entité, *tout attribut constituant une clé* primaire *ne peut pas avoir une valeur nulle*

- Définition : *valeur NULL*
  - Valeur conventionnelle introduite dans une relation pour représenter une information inconnue ou inapplicable

- Définition : *contrainte d'entité*
  - Contrainte d'intégrité imposant que tout attribut participant à une clé primaire soit non nul

Partie 3 : Algèbre relationnel

- Introduction
- Opérateurs ensemblistes
- Opérateurs spécifiques
- Opérations de calcul

Introduction

Algèbre relationnel :

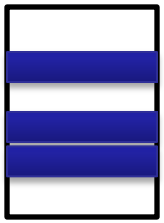
- L'algèbre relationnelle a été inventée par Codd comme une collection d'opérations qui agit sur des relations et qui produit des relations en résultats
- Le langage algébrique est un langage formel qui offre une base théorique solide au modèle relationnel
- L'algèbre relationnelle est à l'origine du langage SQL (section suivante)
- Le résultat d'une opération est une table
- Codd a défini huit opérations, dont certaines peuvent être composées à partir des autres. Ces opérations peuvent être regroupées en deux types d'opérateurs :

- Les opérateurs ensemblistes :
  - union
  - intersection
  - différence
  - produit / division
- Les opérateurs relationnels spécifiques :
  - restriction
  - projection
  - jointure

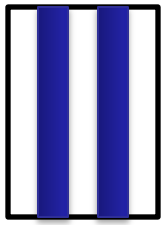
Introduction

- Algèbre relationnel : les huit opérateurs de base

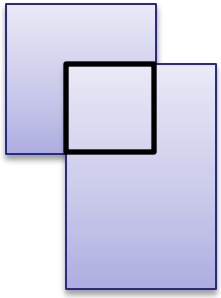
Restriction



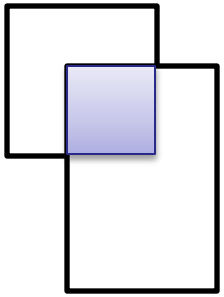
Projection



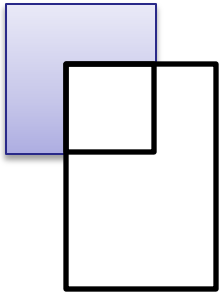
Union



Intersection



Différence



Division

A	X
A	Y
A	Z
B	Y
C	Y

X
Y

A

Produit

A
B
C

X
Y

A	X
A	Y
B	X
B	Y
C	X
C	Y

Jointure

a	x
b	x
c	y

x	1
y	2
z	3

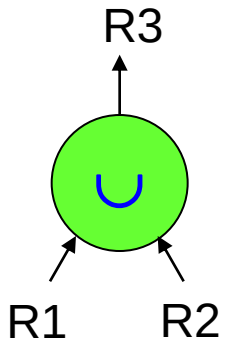
a	x	1
b	x	1
c	y	2

Opérateurs ensemblistes

- Opérateurs ensemblistes - *Union* (∪):

➢ L'union de deux relations R1 et R2 de *même schéma* est une relation R3 contenant des Tuples appartenant à R1 et R2 notée : R3 = (R1 ∪ R2)

La notation graphique



➢ Plusieurs notations ont été introduites pour cette opération :

Exemple : UNION (R1,R2), R1 ∪ R2, APPEND(R1,R2) …

Opérateurs ensemblistes

- Opérateurs ensemblistes - *Union* (∪):

- Exemple :

Patient1

nss	nom	adresse
1807514511	Dupond	Avon
2909112000	Smith	Paris
2873914536	Pierre	Lyon

Patient2

nss	nom	adresse
1807514511	Dupond	Avon
2909112888	Bernard	Paris
2909112899	Eric	Reims
2987812899	Boon	Lille
2873914536	Pierre	Lyon

Patient1 ∪ patient2

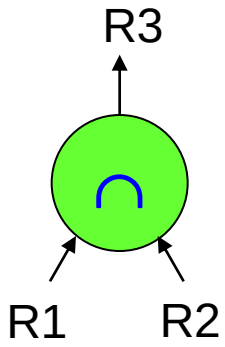
nss	nom	adresse
1807514511	Dupond	Avon
2909112000	Smith	Paris
2873914536	Pierre	Lyon
2909112888	Bernard	Paris
2909112899	Eric	Reims
2987812899	Boon	Lille

Opérateurs ensemblistes

- Opérateurs ensemblistes - *Intersection* (∩):

➢ L'intersection de deux relations R1 et R2 de *même schéma* est une relation R3 contenant des Tuples communs appartenant à R1 et R2 notée : R3 = (R1 ∩ R2)

La notation graphique



➢ Plusieurs notations ont été introduites pour cette opération :

Exemple : INTERSECT (R1,R2), R1 ∩ R2, AND(R1,R2) …

Opérateurs ensemblistes

- Opérateurs ensemblistes - *Union* (∩):

- Exemple :

Patient1

nss	nom	adresse
1807514511	Dupond	Avon
2909112000	Smith	Paris
2873914536	Pierre	Lyon

Patient2

nss	nom	adresse
1807514511	Dupond	Avon
2909112888	Bernard	Paris
2909112899	Eric	Reims
2987812899	Boon	Lille
2873914536	Pierre	Lyon

Patient1 ∩ patient2

nss	nom	adresse
1807514511	Dupond	Avon
2873914536	Pierre	Lyon

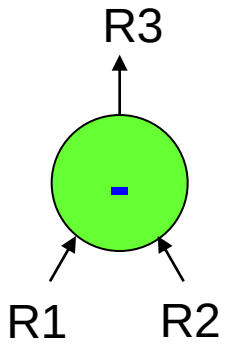
Opérateurs ensemblistes

Opérateurs ensemblistes - *Différence* (-):

- La différence de deux relations R1 et R2 de *même schéma* est une relation R3 contenant des Tuples appartenant à R1 et n'appartenant pas à R2, notée :

  R3 = (R1 - R2)

La notation graphique



- Plusieurs notations ont été introduites pour cette opération :

  Exemple : Différence (R1,R2), R1 - R2, MINUS (R1,R2),…

Opérateurs ensemblistes

- Opérateurs ensemblistes - *Différence* (-):

- Exemple :

Patient1

nss	nom	adresse
1807514511	Dupond	Avon
2909112000	Smith	Paris
2873914536	Pierre	Lyon

Patient2

nss	nom	adresse
1807514511	Dupond	Avon
2909112888	Bernard	Paris
2909112899	Eric	Reims
2987812899	Boon	Lille
2873914536	Pierre	Lyon

Patient2 – patient1

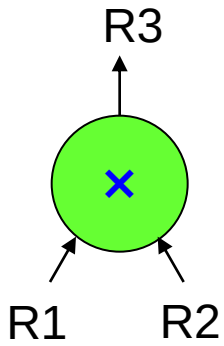
nss	nom	adresse
2909112888	Bernard	Paris
2909112899	Eric	Reims
2987812899	Boon	Lille

Opérateurs ensemblistes

- Opérateurs ensemblistes – *produit cartésien* (×):

➢ Le produit cartésien de deux relations R1 et R2 de *schéma quelconque* est une relation R3 ayant pour attribut la concaténation des attributs des deux relations, et dont les Tuples sont constitués de toutes les concaténations d'un Tuple de R1 à un Tuple de R2 , noté : R3 = (R1 × R2)

La notation graphique



➢ Plusieurs notations ont été introduites pour cette opération :

Exemple : Product (R1,R2), R1 × R2, Times (R1,R2),…

Opérateurs ensemblistes

- Opérateurs ensemblistes – *produit cartésien* (×):

- Exemple :

Patient

nss	Nom_P	adresse
1807514511	Dupond	Avon
2909112000	Smith	Paris
2873914536	Pierre	Lyon

Médecin

N° médecin	Nom_M	service
4125	House	Urgence
5223	Bruno	génécologie

Patient × Médecin

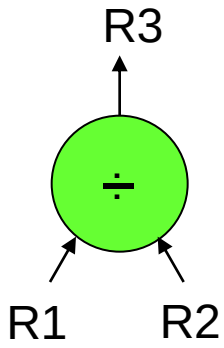
nss	Nom_P	adresse	N° médecin	Nom_M	service
1807514511	Dupond	Avon	4125	House	Urgence
1807514511	Dupond	Avon	5223	Bruno	génécologie
2909112000	Smith	Paris	4125	House	Urgence
2909112000	Smith	Paris	5223	Bruno	génécologie
2873914536	Pierre	Lyon	4125	House	Urgence
2873914536	Pierre	Lyon	5223	Bruno	génécologie

Opérateurs ensemblistes

- Opérateurs ensemblistes – *Division* (÷):

➢ La division de deux relations R1 et R2, *telles que le schéma de R2 est strictement inclus dans celui de R1*, est une relation R3 regroupant toutes les parties d'occurrences de la relation R1 qui sont associées à toutes les occurrences de la relation R2, noté : R3 = (R1 ÷ R2)

La notation graphique



➢ Plusieurs notations ont été introduites pour cette opération :

Exemple : Division(R1,R2), R1 ÷R2

Opérateurs ensemblistes

- Opérateurs ensemblistes – *Division* (÷):

- Exemple :

Consultation

nss	Nom_P	adresse	N° médecin	Nom_M	service
1807514511	Dupond	Avon	4125	House	Urgence
1807514511	Dupond	Avon	5223	Bruno	génécologie
2909112000	Smith	Paris	4125	House	Urgence
2909112000	Smith	Paris	5223	Bruno	génécologie
2873914536	Pierre	Lyon	4125	House	Urgence
2873914536	Pierre	Lyon	4125	Wilson	Urgence
2873914536	Pierre	Lyon	5223	Bruno	génécologie

Patient

nss	Nom_P	adresse
1807514511	Dupond	Avon
2909112000	Smith	Paris
2873914536	Pierre	Lyon

Consultation ÷ Patient

N° médecin	Nom_M	service
4125	House	Urgence
5223	Bruno	génécologie

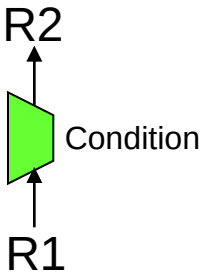
Exemple de division :
La relation résultat contient donc tous les médecins de la relation Consultation qui auscultent tous les patients de la relation Patient

Opérateurs spécifiques

- Opérateurs spécifiques - *restriction* (σ):

  - La restriction (ou sélection) de la relation R par une qualification Q est une relation R' de même schéma dont les Tuples sont ceux de R satisfaisant la qualification Q
  - La qualification Q peut être exprimée à l'aide de constantes, comparateurs arithmétiques (>,>= , < ,>= , =, ≠) et des opérateurs logiques (∩,∪, ¬)
  - notée : σ Condition(R)

La notation graphique



  - Plusieurs notations ont été introduites pour cette opération :

    Exemple : RESTRICT(R1, Condition)

Opérateurs ensemblistes

- Opérateurs spécifiques - *restriction* (σ):

- Exemple :

Patient

nss	nom	adresse
1807514511	Dupond	Avon
2909112888	Bernard	Paris
2909112899	Eric	Paris
2987812899	Boon	Paris
2873914536	Pierre	Lyon

Patient-Paris = Patient[adresse=”Paris”]

Patient-Paris

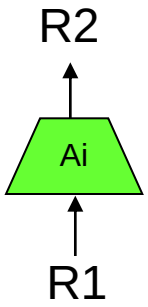
nss	nom	adresse
2909112888	Bernard	Paris
2909112899	Eric	Paris
2987812899	Boon	Paris

Opérateurs spécifiques

Opérateurs spécifiques - *projection* (Π):

- La projection d'une relation R sur les attributs A_i est une relation R', dont les Tuples sont obtenus par élimination des attributs de R autre que A_i et par suppression des doublons. notée : $\Pi\ A_i(R)$

La notation graphique



- Plusieurs notations ont été introduites pour cette opération :

  Exemple : PROJECT(R, liste des attributs), R[liste des attributs]

Opérateurs ensemblistes

- Opérateurs spécifiques - *projection* (Π):

- Exemple :

Patient

nss	nom	adresse
1807514511	Dupond	Avon
2909112888	Bernard	Paris
2909112899	Eric	Paris
2987812899	Boon	Paris
2873914536	Pierre	Lyon

adresse = Patient[adresse]

adresse

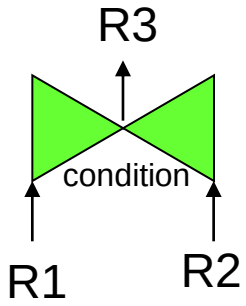
adresse
Avon
Paris
Lyon

Opérateurs spécifiques

- Opérateurs spécifiques - *Jointure* (⋈):

  - La jointure de deux relations R1 et R2 selon une qualification Q est l'ensemble du produit cartésien R1 x R2 satisfaisant la qualification Q (*condition de rapprochement*)
  - Notation : R3= R1⋈R2
  - Dans le cas de la *jointure naturelle* la condition est simplement omise

La notation graphique



  - Plusieurs notations ont été introduites pour cette opération :

    Exemple : JOIN(R1,R2, condition)

Opérateurs ensemblistes

- Opérateurs spécifiques - *Jointure* (⋈):

- Exemple jointure naturelle (sans condition):

Patient

nss	Nom_P	N° médecin	adresse
1807514511	Dupond	4125	Avon
2909112000	Smith	5223	Paris
2873914536	Pierre	4125	Lyon

Médecin

N° médecin	Nom_M	service
4125	House	Urgence
5223	Bruno	génécologie

R = JOIN(Patient, Médecin)

R

nss	Nom_P	adresse	N° médecin	Nom_M	service
1807514511	Dupond	Avon	4125	House	Urgence
2909112000	Smith	Paris	5223	Bruno	génécologie
2873914536	Pierre	Lyon	4125	House	Urgence

Opérateurs spécifiques

- Opérateurs spécifiques - *Jointure* ($\bowtie$) - suite:

- La condition de rapprochement peut être exprimée à l'aide de constantes, comparateurs arithmétiques (>,>= , < ,>= , =, ≠) et des opérateurs logiques ($\cap$,$\cup$, $\neg$)

- Selon le type d'opérateur nous distinguons :
  - L'*équijointure* dans le cas ou l'opérateur est l'égalité,
  - L'*inéqui-jointure* dans le cas ou l'opérateur est l'un des opérateur suivant { <,≤,>, ≥, ≠}

- *Auto jointure* : est une jointure d'une table avec elle-même(tout se passe comme si on avait deux copies différentes de la même table)

- *Jointure externe* ($\overset{\bullet}{\bowtie}$) : identique à la jointure mais en complétant les attributs vide par des nulles

Opérateurs ensemblistes

- Opérateurs spécifiques - *Jointure* (⋈):

- Exemple inéqui-jointure :

Patient

nss	Nom_P	N° médecin	adresse
1807514511	Dupond	4125	Avon
2909112000	Smith	5223	Paris
2873914536	Pierre	4125	Lyon

Médecin

N° médecin	Nom_M	service
4125	House	Urgence
5223	Bruno	génécologie

R = JOIN(Patient, Médecin)
Condition : nss >2000000000

R

nss	Nom_P	adresse	N° médecin	Nom_M	service
2909112000	Smith	Paris	5223	Bruno	génécologie
2873914536	Pierre	Lyon	4125	House	Urgence

Opérateurs ensemblistes

- Opérateurs spécifiques - *Jointure* externe (⟕):
- Exemple jointure externe (*outer joint*) :

Patient

nss	Nom_P	N° médecin	adresse
1807514511	Dupond	4125	Avon
2909112000	Smith	5223	Paris
2873914536	Pierre	4100	Lyon

Médecin

N° médecin	Nom_M	service
4125	House	Urgence
2321	Wilson	Urgence
5223	Bruno	génécologie

R = Ext-JOIN(Patient, Médecin)

R

nss	Nom_P	adresse	N° médecin	Nom_M	service
1807514511	Dupond	Avon	4125	House	Urgence
2909112000	Smith	Paris	5223	Bruno	génécologie
2873914536	Pierre	Lyon	4100	-	-
-	-	-	5223	Bruno	génécologie

Opérateurs de bases

- Opérateurs de bases

➢ Les six opérations fondamentales sont : l'union, la différence, le produit, la restriction, la projection et la jointure.

➢ Les autres opérations peuvent se déduire aisément à partir de ces opérations fondamentales

- Exemple :

$$R1 \underset{condition}{\bowtie} R2 \quad \Leftrightarrow \quad \sigma_{condition}(R1 \times R2)$$

$$R1 \cap R2 \quad \Leftrightarrow \quad R1 - (R1 - R2)$$

Opérateurs de bases

- Composition d'opérateurs :

➢ Un programme d'opérations de l'algèbre relationnelle peut être représentée par un arbre ou les nœuds correspondent aux représentations graphiques des opérations

- Exemple : donner les Noms et Adresses des buveurs ayant bu plus de 10 bouteilles de CHABLIS 1976 et le degré de ce vin

Buveur(Nom, Prénom, Adresse)

Vin(Cru, Millésime, Qualité, Quantité, Degré)

Abus(Buveur, Vin, Date, Quantité)

Opérateurs de bases

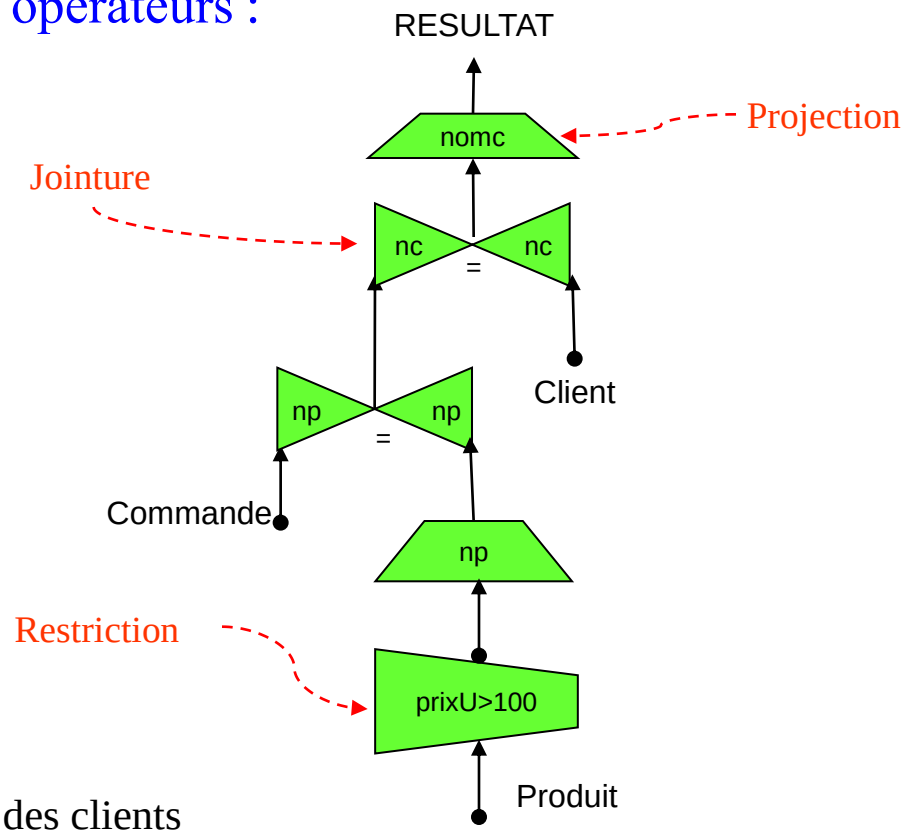
- Composition d'opérateurs :

- Exemple 1
  - *Produit* (*np*, *désignation*, *prixU*)
  - *Client*(*nc*, *nomc*)
  - *Fournisseur*(*nf*, *nomf*)
  - *Commander* (*np*, *nc*, *nf*, *date*, *qte*)

Question : Donner les noms des clients qui ont commandé au moins un produit de prix supérieure à 100 euros?

Opérateurs de bases

- Composition d'opérateurs :



Question : Donner les noms des clients qui ont commandé au moins un produit de prix supérieure à 100 euros

Opérateurs de bases

- Composition d'opérateurs :

- Exemple 2 : donner les Noms et Adresses des buveurs ayant bu plus de 10 bouteilles de CHABLIS 1976 et le degré de ce vin

Buveur(Nom, Prénom, Adresse)

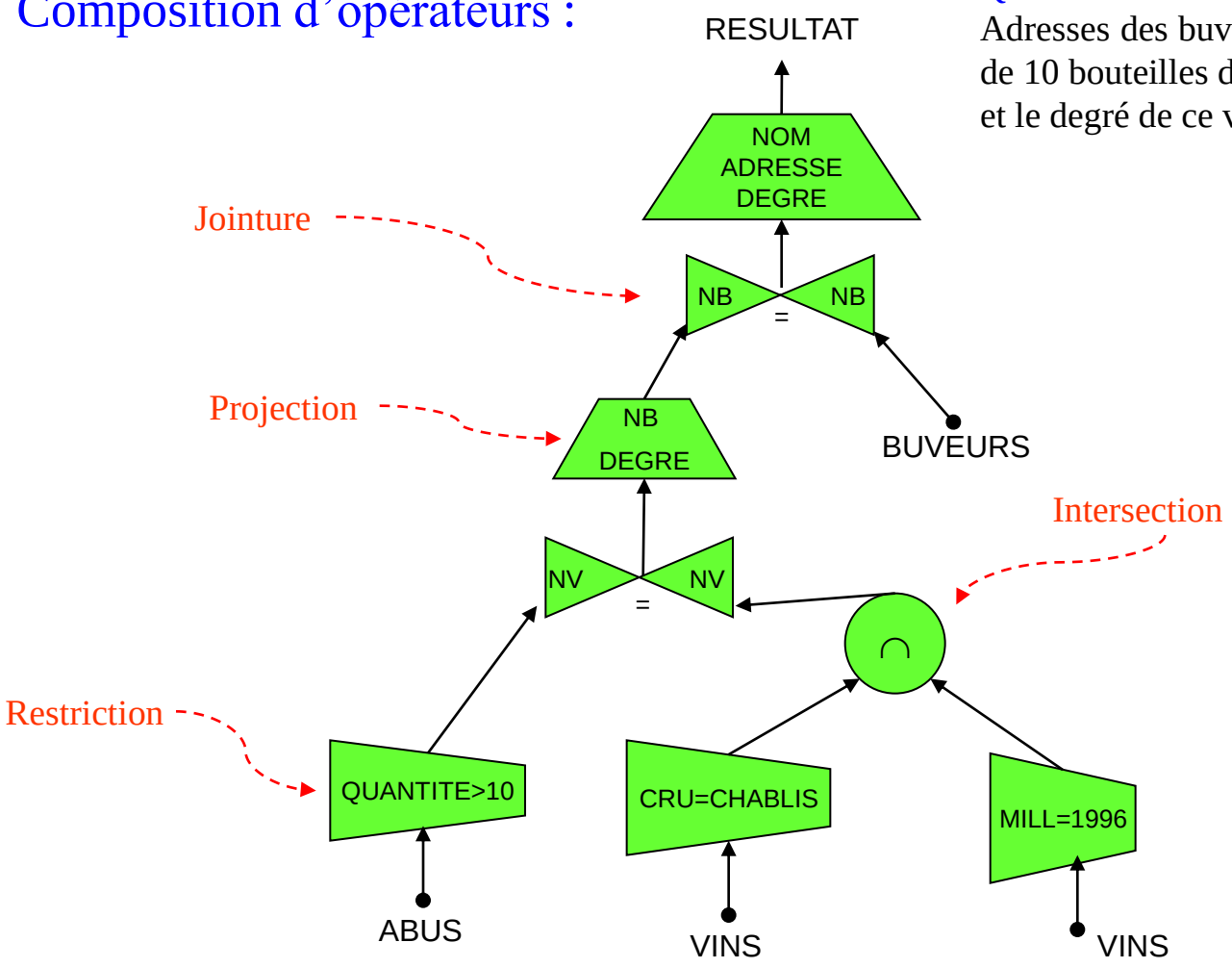
Vin(Cru, Millésime, Qualité, Quantité, Degré)

Abus(Buveur, Vin, Date, Quantité)

Opérateurs de bases

- Composition d'opérateurs :

Question : donner les Noms et Adresses des buveurs ayant bu plus de 10 bouteilles de CHABLIS 1976 et le degré de ce vin



Partie 4 : le langage SQL

- Introduction
- Norme
- Définition des données
- Manipulation des données

Introduction

- Le langage SQL :(*Structured Query Language*)

- Définition : c'est un langage permettant la définition, la manipulation et le contrôle des bases de données

✓ Il est se base sur l'algèbre relationnel

✓ En 1977, création du langage SEQUEL (*Structured English Query Language*)

✓ SEQUEL continue de s'enrichir pour devenir SQL

✓ IBM sort sa 1ère version SQL en 1981

✓ En 1986, la première norme SQL (SQL-1) de l'ISO (International Standard Organisation) apparaît

✓ SQL est actuellement implémenté par les principaux éditeurs de SGBDR (Oracle, IBM DB2, Microsoft SQL Server, MySQL, Sybase, etc.)

Norme

- Le langage SQL :(*Structured Query Language*)

- SQL86
  - version préliminaire

- SQL89 (SQL, SQL1)
  - niveau minimal supporté

- SQL92 (SQL2)
  - Support accru de l ’intégrité
  - Le standard le plus répandu

- SQL-99 (SQL3)
  - Extensions objet (UDT),
  - requêtes récursives, déclencheurs,…

- SQL:2003 :
  - introduction de fonctions pour la manipulation XML,…

Définition

- Le langage SQL :(*Structured Query Language*)

- Remarques :
  - ✓ SQL est un langage de requête, pas un langage de programmation
  - ✓ Une *instruction* SQL peut s'écrire sur plusieurs lignes. Pour être exécutée, l'instruction doit se terminer par un point-virgule
  - ✓ Il n'y pas de variables ni de structures de contrôles (if, while, for)
  - ✓ SQL est intégré dans les langages de programmation comme PL/SQL, C ou Java

Définition

- Le langage SQL :(*Structured Query Language*)

Terminologie :

- ✓ Quelque terme change entre algèbre relationnelle et SQL

Terme relationnel formel	SQL
Relation	Table
Tuple	Ligne / enregistrement
Attribut	Colonne / champ
Clé primaire	Primary Key
Ensemble de schémas	Catalogue

Type des données

- Dans les normes de SQL nous distinguons plusieurs types de données, quelques exemples:

Type	Intitulé	Taille
INTEGER	entiers signés	4 octets
BIGINT	entiers signés	8 octets
REAL	réels	4 octets
DOUBLE PRECISION	réels	8 octets
NUMERIC [(précision, [longueur])]	données entières et réelles	
CHAR(longueur)	chaînes de caractères de longueur fixe	Long <255
VARCHAR(longueur)	chaînes de caractères de longueur variable	Long <2000
DATE	date	
TIMESTAMP	date + heure	
MONEY	valeurs monétaires	
TEXT	chaînes de caractères de longueur variable	
BOOLEAN	valeurs Booléenne	

Type des données

- Exemple :

Type	Exemple
INTEGER	2, 3, 459
REAL	3.27E-4, 24E5
DOUBLE PRECISION	3.27265378426E-4, 24E12
NUMERIC [p, l] , DEC[p,l]	2.5, 456.342, 6
DATE	'1998-08-25'
TIMESTAMP	'1998-08-25 14:04:32.25'

Catégories d'instructions SQL

- Les instructions SQL sont regroupées en catégories en fonction de leur utilité et des entités manipulées. Nous pouvons distinguer cinq catégories, qui permettent :

- La définition des éléments d'une base de données (tables, colonnes, clefs, index, contraintes, . . .)
- La manipulation des données (insertion, suppression, modification, …)
- La gestion des droits d'accès aux données (acquisition et révocation des droits)
- La gestion des transactions
- Le SQL intégré

Catégories d'instructions SQL

- SQL est alors décomposé en 5 parties :

1. LDD (langage de définition des données) : permet de modifier la structure de la base de données
2. LMD (langage de manipulation des données) : permet de consulter / modifier le contenu de la base de données
3. LCD (langage de contrôle des données) : permet de gérer les privilèges, c'est-à-dire les utilisateurs et les actions qu'ils peuvent entreprendre
4. LCT (langage de contrôle des transactions) : permet de gérer les transactions, c'est-à-dire rendre atomique divers ordres enchaînés en séquence
5. *SQL procedural* : est un ensemble d'outils pour que SQL s'interface avec des langages hôtes

Langage de définition des données

- LDD (langage de définition des données) : permet de modifier la structure de la base de données

➢ Les instructions du LDD sont : CREATE, ALTER, DROP, RENAME, AUDIT, NOAUDIT, ANALYZE, TRUNCATE

- Plan :

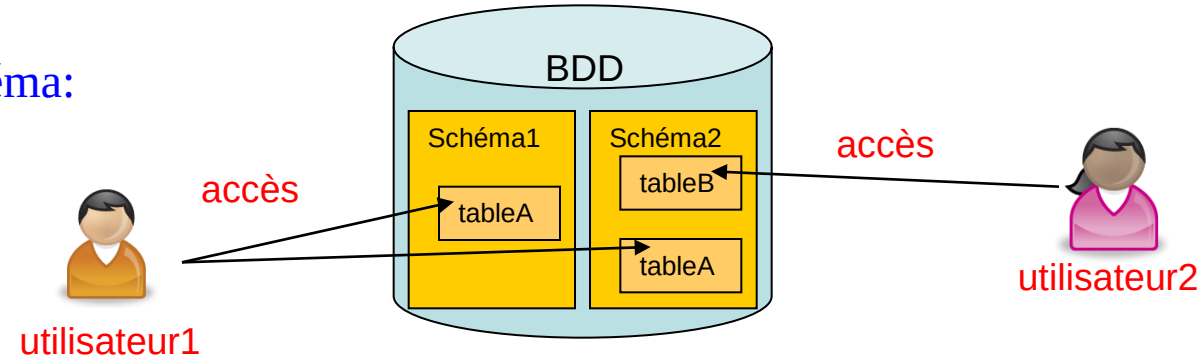
- Les objets : table, schéma, vues
- Création des objets
- Modification des objets
- Suppression

LDD

- Plan :
  - Les objets : table, schéma, vues
  - Création des objets
  - Modification des objets
  - Suppression

LDD - objet

- Schéma:



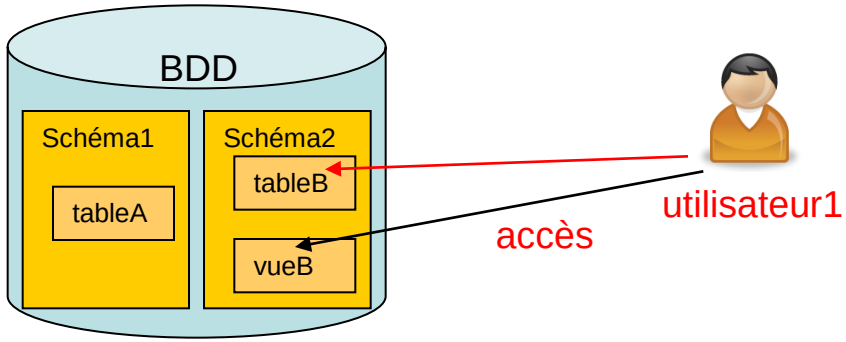
- Un schéma est une collection d'objets comme des tables ou des vues, associée à un nom d'utilisateur de la base de données
- Dans la pratique, tout utilisateur créant un objet dans une base de données, engendre systématiquement un schéma
- L'accès aux objets d'un schéma par un utilisateur quelconque, s'effectue par l'intermédiaire du nom du propriétaire associé à celui de l'objet :

  nom_utilisateur. *nom_objet*

- Chaque objet dans une base de données créé par un utilisateur est ainsi clairement identifié par le couple *propriétaire.objet*, évitant d'une part d'éventuels noms doubles et d'autre part, des problèmes de sécurité

LDD - objet

- Vues (VIEW):



- Une vue est une relation virtuelle au sens où ses instances n'existent pas physiquement mais sont calculées dynamiquement à chaque invocation de la vue
- Une vue est définie par une requête 'SELECT' qui utilise des relations (tables) ou des vues existantes
- A la différence d'une table, une vue ne consomme pas d'espace de stockage physique
- Les vues peuvent servir à recueillir des données régulièrement consultées et mises à jour
- Les vues sont souvent employées pour des raisons de sécurité (il possible de ne montrer qu'une partie des données d'une table)

LDD

- Plan :
  - Les objets : table, schéma, vues
  - Création des objets
  - Modification des objets
  - Suppression

LDD - création d'objet

- Création d'un objet(*CREATE*) :

- La Syntaxe pour la création d'une table en SQL est la suivante :

```
CREATE TABLE nom_table (
        attribut1 type1 [DEFAULT valeur_par_défaut] [contrainte sur attribut1],
        attribut2 type2 [DEFAULT valeur_par_défaut] [contrainte sur attribut2],
        …
        [CONSTRAINT contrainte1 sur relation],
        ….
        [CONSTRAINT contrainteN sur relation)];
```

- Les éléments entre [ ] sont optionnels

- Chaque table est définie par : un nom, des d'attributs et des contraintes

- Chaque attribut est défini par : un type, une valeur par défaut et des contraintes sur attribut

LDD – création d'objet

- Création d'un objet(*CREATE*) – Exemple simple :

➢ On veut créer la table personne contenant pour chaque personne un ID, un prénom, un nom et l'âge

```
CREATE TABLE Personne(
            ID CHAR(8),
            prénom  CHAR (20),
            nom CHAR (30))
```

➢ La table Personne existe physiquement mais elle est encore vide de données

LDD – création d'objet

- Création d'un objet(*CREATE*) :

➢ Les contraintes possibles sur les attributs:

```
PRIMARY KEY
NOT NULL
UNIQUE
REFERENCES nom_de_relation2      //pour les clés étrangères
CHECK (expression logique)
```

➢ S'il y a plusieurs contraintes sur un attribut, elles sont séparées par des virgules

- Exemple :

```
//l'âge d'une personne ne peut pas être négatif et ne dépasse pas 150 ans
CHECK (age>=0 and age<=150)
```

LDD - contraintes

- Règles d'usage des contraintes :

  - Un attribut ne peut pas être NULL et NOT NULL en même temps
  - UNIQUE et PRIMARY KEY doivent être spécifiés qu'avec des attributs NOT NULL
  - CHECK ne peut contenir que des prédicats de comparaison d'attributs avec des constantes

- Exemple :

```
age INTEGER CHECK (age>=0 and age<=150),
NuméroSS INTEGER NOT NULL UNIQUE,
```

LDD – création d'objet

- Création d'un objet(*CREATE*) :

- Les contraintes possibles sur les relations (plusieurs attributs):

```
UNIQUE (liste d'attributs)
PRIMARY KEY (liste d'attributs)
//clé étrangère composé de plusieurs attributs
FOREIGN KEY (liste d'attributs) REFERENCES nom_de_relation2 (liste d'attributs)
```

- S'il y a plusieurs contraintes, elles sont séparées par des virgules

- Exemple :

```
//le nom et le prénom doivent être unique dans la table
CONSTRAINT maContraineNom UNIQUE (nom, prénom),
CONSTRAINT maContrainePrim PRIMARY KEY (nom, Date)
```

LDD - contraintes

- Règles d'usage des contraintes :

  - FOREIGN KEY et REFERENCES doivent apparaître simultanément
  - Les attributs de FOREIGN KEY doivent correspondre à des attributs déclarés comme PRIMARY KEY ou UNIQUE dans une autre table
  - Si les attributs de FOREIGN KEY correspondent à une clé primaire dans une autre table, la clause REFERENCES ne spécifie que le nom de cette dernière
  - Si les attributs de FOREIGN KEY correspondent à une clé candidate dans une autre relation, la clause REFERENCES doit mentionner le nom de cette table ainsi que la liste des attributs composant la clé candidate

- Exemple :

```
CONSTRAINT C1 PRIMARY KEY (num, numPersonne),
CONSTRAINT C2 FOREIGN KEY (num) REFERENCES VOITURE,
CONSTRAINT C2 FOREIGN KEY (nom, prénom) REFERENCES Personne(nom, prénom)
```

LDD -Exemple

- Création des tables – Exemple 1 :
- La requête SQL suivante crée la table VOITURE :

Voiture

Immatriculation	Marque	couleur	Prix	Puissance

```
CREATE TABLE VOITURE(
    Immatriculation VARCHAR(8),
    Marque VARCHAR(20),
    Couleur VARCHAR(15),
    Prix INTEGER CONSTRAINT C1 CHECK (Prix>=0),
    Puissance INTEGER,
    CONSTRAINT C2 PRIMARY KEY (Immatriculation) );
```

LDD - Exemple

- Création des tables – Exemple 2 :
- La requête SQL suivante crée la table ACHAT :

Achat

NumImmatriculation	NumPersonne	dateAchat

```
CREATE TABLE ACHAT(
    numImmatriculation VARCHAR(8),
    numPersonne INTEGER,
    dateAchat DATE,
    CONSTRAINT C1 PRIMARY KEY (numImmatriculation, numPersonne),
    CONSTRAINT C2 FOREIGN KEY (numImmatriculation) REFERENCES
    VOITURE(Immatriculation),
    CONSTRAINT C3 FOREIGN KEY (numPersonne) REFERENCES
    Personne(NuméroSS) );
```

LDD - VIEW

- VUES:

- La création d'une vue se fait grâce à la clause CREATE VIEW suive :
  - du nom que nous donnons à la vue,
  - du nom des colonnes dont nous désirons afficher dans cette vue,
  - d'une clause AS précédant la sélection

- La syntaxe de création d'une vue est la suivante :

```
CREATE VIEW nom_de_la_vue (colonneA, colonneB, colonneC,
colonneD)
AS <clause SELECT>;
```

```
CREATE OR REPLACE VIEW nom_de_la_vue (colonneA, colonneB,
colonneC, colonneD) AS <clause SELECT>;
```

LDD - VIEW

- VUES:

- Après sa création, une vue peut être utilisée de la même façon qu'une table
- Des requêtes de sélection ou de manipulation de données peuvent être appliquées à la vue
- Les vues sont calculées, à partir des tables, à chaque nouvelle réutilisation
- La vue représente une sorte d'intermédiaire entre la BDD et l'utilisateur. Cela a de nombreuses conséquences:
  - une sélection des données à afficher (simplification des requêtes),
  - une restriction d'accès à la table pour l'utilisateur, c'est-à-dire une sécurité des données accrue,
  - un regroupement d'informations au sein d'une entité (intégrité des données),
  - Transparence pour l'utilisateur il les utilise comme des tables

LDD - VIEW

- VUES - Exemples:

- Exemple 1: la requête suivante nous permet de créer la vue VOITURES_CHERES, cette vue regroupe tous les tuples de la table VOITURE dont le prix est supérieur à 5 000 € :

```
CREATE VIEW VOITURES_CHERES
AS SELECT * FROM VOITURE WHERE Prix > 5 000;
```

- Exemple 2 : la requête suivante nous permet de créer la vue VOITURES_ PUISSANTES mais avec des noms d'attributs différents de celui de la table VOITURE :

```
CREATE VIEW VOITURES_PUISSANTES (immat_voiture, marque_voiture)
AS SELECT Immatriculation, Marque FROM VOITURE WHERE Puissance
> 6;
```

LDD - schéma

- Schéma:

- Il est possible de créer explicitement un schéma à l'aide de la commande CREATE SCHEMA :

```
CREATE SCHEMA AUTHORIZATION nom_schéma{
  instruction_CREATE_TABLE | instrustion_CREATE_VIEW | instruction_GRANT
}
```

- La commande CREATE SCHEMA propose un moyen de créer des tables, des vues et d'accorder des privilèges pour les objets à partir d'une unique instruction

- GRANT : privilèges d'accès

- Remarques :

```
//Permet de connaître le schéma d'une table créée précédemment (sous Oracle) :
DESC nom_schéma
// sous MySQL :
SHOW nom_schéma
```

LDD – création d'un schéma

- Création des schémas – Exemple :
- La requête SQL suivante crée le schéma USER1 :

```
CREATE SCHEMA AUTHORIZATION USER1
        CREATE TABLE table1(
                num INTEGER PRIMARY KEY,
                nom VARCHAR(30),
                prénom VARCHAR(30))

        CREATE VIEW vue1 AS
                SELECT nom, prénom
                FROM table1 WHERE num >= 200

        GRANT SELECT ON vue1 TO user20;
```

LDD

- Plan :
  - Les objets : table, schéma, vues
  - Création des objets
  - Modification des objets
  - Suppression

LDD - modification

- Modification d'un objet table :

➢ La Syntaxe pour la mise à jour d'une table en SQL est la suivante :

```
ALTER TABLE nom_table  opération;
```

➢ L'opération peut être :

```
// Ajout ou modification d'un attribut
        ADD/MODIFY COLUMN attribute_name data_type

// Suppression d'un attribut
        DROP COLUMN attribute_name

//Modification d'une valeur par défaut
        ALTER COLUMN attribute_name SET DEFAULT value

        //Renommer un attribut
        RENAME COLUMN old_attribute_name TO new_attribute_name

// Renommer une table
        RENAME TO new_table_name
```

LDD - modification

- Modification d'un objet table - suite :

➢ L'opération peut être :

```
// Suppression d'une contrainte
        DROP CONSTRAINT constraint_name
```

```
// Ajout d'une contrainte
        ADD CHECK (attribute_name condition);
        ADD CONSTRAINT constraint_name UNIQUE (attribute_name);
        ADD FOREIGN KEY (attribute_name) REFERENCES table_name
        (attribute_name);
```

- Modification d'un objet vue :

```
// Il est possible de renommer une vue
        RENAME nom_vue TO nouveau_nom
```

LDD

- Plan :
  - Les objets : table, schéma, vues
  - Création des objets
  - Modification des objets
  - Suppression

LDD - suppression

- Suppression d'un objet(*DELETE*) :

➢ La Syntaxe pour la suppression d'une table en SQL est la suivante :

```
DROP TABLE nom_table;
DROP TABLE nom_table CASCADE;
```

- Avant de supprimer une table, il faut supprimer toutes les références à cette table dans les autres tables (ou utiliser le CASCADE CONSTRAINTS)
- Avant de supprimer une table, il faut avoir redéfini ou supprimé toutes les vues impliquant cette table (nous reviendrons un peu plus loin sur la définition des vues)
- La suppression d'une table implique de redéfinir ou supprimer toutes les requêtes impliquant la table supprimée.

➢ La Syntaxe pour la suppression d'une vue en SQL est la suivante :

```
DROP VIEW nom_vue [CASCADE | RESTRICT];
```

➢ La Syntaxe pour la suppression d'un schéma en SQL est la suivante :

```
DROP SHEMA nom_shema [CASCADE | RESTRICT];
```

Langage de manipulation des données

- LMD (langage de manipulation des données) : permet de consulter / modifier le contenu de la base de données

- Les instructions du LMD sont : INSERT, SELECT, UPDATE et DELETE

- Plan :
  - Insertion
  - Affichage / sélection
  - Mise à jour
  - Suppression

LMD

- Plan :
  - Insertion
  - Affichage
  - Mise à jour
  - Suppression

LMD - Insertion

- Le commande INSERT permet d'ajouter un ou plusieurs Tuples à une table

- La syntaxe de la primitive INSERT en SQL est la suivante :

```
INSERT INTO   nom_table  [attribut1, attribut2...]
VALUES (val1, val2,...) | <clause SELECT>;
```

- Il est possible de ne pas énumérer les noms de colonnes si l'ajout de données concerne un enregistrement complet de la table

- Il est également possible d'insérer des données provenant d'une autre table par l'intermédiaire d'une commande *select*

- Les éléments entre [ ] sont optionnels et le symbole "|" signifie "ou"

LMD - Insertion

- Le commande INSERT – Exemple :

➢ Quelques exemples de requêtes d'insertion de données dans une table :

```
INSERT INTO Voiture values('123ABC75', 'Renault', 'Rouge', 7, 10 000);
INSERT INTO Voiture values('123ABC92', 'Peugeot', 'Noire', 5, 2 500);
INSERT INTO Voiture values('123ATT77', 'Rover', 'Bleue', 6, 13 000);
INSERT INTO Voiture values('698KLM45', 'Citroën', 'Verte', 5,  5 000);
INSERT INTO Voiture values('956BZD59', 'Peugeot', 'Jaune', 5, 16 000);
INSERT INTO Voiture values('156AMN86', 'Renault', 'Bleue',  7, 5 250);
```

LMD

- Plan :
  - Insertion
  - Affichage
    - Syntaxe
    - Paramètres
    - Prédicats
    - Fonctions statistiques
    - Opérateurs relationnels
    - Jointure
  - Mise à jour
  - Suppression

LMD - Affichage

- Le commande SELECT permet de sélectionner plusieurs Tuples d'une ou plusieurs table

- La syntaxe de la primitive SELECT en SQL est la suivante :

```
SELECT * | [DISTINCT | ALL] <attributs>
FROM <tables>
[WHERE <condition logique de niveau Tuple>]
[GROUP BY <attributs>[HAVING <condition logique de niveau groupe>]]
[ORDER BY <attributs>];
```

- Les éléments entre [ ] sont optionnels et le symbole "|" signifie "ou"

LMD - Affichage

- Paramètres :

- DISTINCT et ALL : permettent respectivement d'éliminer ou de conserver les doublons après une projection mise en œuvre par SELECT
- FROM : désigne les tables concernées par la requête
- WHERE : spécifie les critères de sélection (optionnelle) qui peuvent être des prédicats (conditions) de restriction ou de jointure
- GROUP BY : partitionne la relation en groupes (en sous ensemble)
  - HAVING : lorsque les critères portent sur des fonctions de groupe
- ORDER BY : permet de trier les résultats obtenus selon un tri ascendant (ASC) ou descendant (DESC)

Remarque : nous pouvons également insérer dans la clause WHERE, une nouvelle instruction SELECT (requête imbriquée) ce qui permet de travailler sur un ensemble de Tuples restreints par cette sélection

LMD - Affichage

- Affichage - Exemples :

➢ Sélection simple sans conditions (tous les Tuples):

```
SELECT * FROM VOITURE;
```

➢ Sélection d'une colonne :

```
SELECT Marque FROM VOITURE;
```

- Cette requête risque d'afficher des doublons

➢ Sélection d'une colonne sans doublons :

```
SELECT DISTINCT Marque FROM VOITURE;
```

LMD - Affichage

- Affichage – Condition (la clause WHERE):

➢ WHERE : spécifie une condition de sélection

– Une condition de sélection définit un critère qui, appliqué à un Tuple, est vrai, faux ou inconnu

– Cette condition peut inclure des opérateurs booléens (AND, OR, NOT), des conditions élémentaires et des parenthèses

– Les opérateurs logiques peuvent être combinés entre eux en respectant un ordre de priorité (NOT, AND et OR)

LMD - Affichage

Affichage – Prédicats :

- Prédicat (condition) : permet de comparer 2 expressions de valeurs :
  - La première expression contenant des spécifications de colonne est appelée terme
  - La seconde expression contenant seulement des spécifications de constants est appelée constante
- Il existe plusieurs types de prédicats :
  - De comparaison : =, !=, >, ≥, <, ≤ (valables pour INTEGER, FLOAT, CHAR, VARCHAR et DATE)
  - D'intervalle : permet de tester si la valeur d'un terme est comprise entre la valeur de 2 constantes (BETWEEN)
  - De comparaison de texte : permettant de tester si un terme de type chaîne de caractères contient une ou plusieurs sous-chaînes (LIKE)
    - Une utilisation de jokers est possible : le % remplace un nombre quelconque de caractères (0 compris) tandis que le _ remplace un caractère unique.

LMD - Affichage

Affichage – Prédicats - suite :

- Il existe plusieurs types de prédicats - suite :
  - De test de nullité : qui permet de tester si un terme a une valeur convenue nulle, signifiant que sa valeur est inconnue ou que le champ n'a pas été renseigné (IS NULL)
  - D'appartenance : qui permet de tester si la valeur d'un terme appartient à une liste de valeurs constantes (IN)

LMD - Affichage

- Affichage - Exemples :

➢ Quelques exemple de requêtes sur la table Voiture :

Voiture

Immatriculation	Marque	Couleur	Puissance	Prix
123ABC75	Renault	Rouge	7	10 000
123ABC92	Peugeot	Noire	5	2 500
123ATT77	Rover	Bleue	6	13 000
698KLM45	Citroën	Verte	5	5 000
956BZD59	Peugeot	Jaune	5	16 000
156AMN86	Renault	Bleue	7	5 250

LMD - Affichage

- Affichage - Exemples :

- Question : afficher les Tuples de la table VOITURE dont la puissance est inférieure ou égale à 6 chevaux ou dont le prix est supérieur ou égal à 10 000 €

- Réponse :

```
SELECT * FROM VOITURE
WHERE Puissance <= 6 OR Prix >= 10 000;
```

- Affichage

Voiture

Immatriculation	Marque	Couleur	Puissance	Prix
123ABC75	Renault	Rouge	7	10 000
123ABC92	Peugeot	Noire	5	2 500
123ATT77	Rover	Bleue	6	13 000
698KLM45	Citroën	Verte	5	5 000
956BZD59	Peugeot	Jaune	5	16 000
156AMN86	Renault	Bleue	7	5 250

LMD - Affichage

- Affichage - Exemples :

- Question : afficher les Tuples de la table VOITURE dont la puissance n'est pas égale à 6 chevaux

- Réponse :

```
SELECT * FROM VOITURE
WHERE NOT (Puissance= 6);
```

- Affichage

Voiture

Immatriculation	Marque	Couleur	Puissance	Prix
123ABC75	Renault	Rouge	7	10 000
123ABC92	Peugeot	Noire	5	2 500
123ATT77	Rover	Bleue	6	13 000
698KLM45	Citroën	Verte	5	5 000
956BZD59	Peugeot	Jaune	5	16 000
156AMN86	Renault	Bleue	7	5250

LMD - Affichage

- Affichage - Exemples :

- Question : afficher les Tuples de la table VOITURE dont le prix est compris entre 2000 et 6000 euros

- Réponse :

```
SELECT * FROM VOITURE
WHERE Prix BETWEEN 2 000 AND 6 000;
```

- Affichage

Voiture

Immatriculation	Marque	Couleur	Puissance	Prix
123ABC75	Renault	Rouge	7	10 000
123ABC92	Peugeot	Noire	5	2 500
123ATT77	Rover	Bleue	6	13 000
698KLM45	Citroën	Verte	5	5 000
956BZD59	Peugeot	Jaune	5	16 000
156AMN86	Renault	Bleue	7	5 250

LMD - Affichage

- Affichage - Exemples :

- Question : afficher les Tuples de la table VOITURE dont la marque commence par 'Ren' dont la couleur est noire, bleue ou jaune et dont le prix est connu

- Réponse :

```
SELECT * FROM VOITURE
WHERE Marque LIKE 'Ren%' AND Couleur IN
('Noire', 'Bleue', 'Jaune') AND Prix IS NOT NULL;
```

- Affichage

Voiture

Immatriculation	Marque	Couleur	Puissance	Prix
123ABC75	Renault	Rouge	7	10 000
123ABC92	Peugeot	Noire	5	2 500
123ATT77	Rover	Bleue	6	13 000
698KLM45	Citroën	Verte	5	5 000
956BZD59	Peugeot	Jaune	5	16 000
156AMN86	Renault	Bleue	7	5 250

LMD – fonctions statistiques

- Affichage – Fonctions statistiques :

- Les fonctions statistiques sont appliquées à l'ensemble d'une colonne (ou d'un groupe) et fournissent une valeur unique Les fonctions statistiques

- Les fonctions statistiques peuvent être utilisées dans les clauses SELECT, WHERE et HAVING

- Il existe plusieurs fonctions statistiques :
  - MAX : maximum
  - MIN : minimum
  - SUM : somme
  - AVG : moyenne
  - COUNT : nombre de lignes

LMD – fonctions statistiques

- Affichage – Fonctions statistiques :

- Les fonctions MAX et MIN renvoient respectivement le maximum et le minimum d'un champ (utilisables pour les types CHAR, VARCHAR, DATE et NUMBER)

- La fonction SUM effectue, pour un ensemble de Tuples, la somme des valeurs d'un attribut (utilisable uniquement pour le type NUMBER)

- La fonction COUNT comptabilise le nombre de lignes pour lesquelles l'attribut est non NULL

- La fonction AVG calcule, pour un ensemble de Tuples, la moyenne arithmétique des valeurs d'un attribut (utilisable uniquement pour le type NUMBER)

  - Elle vérifie la formule suivante :

    AVG = Somme des valeurs non NULL / nombre de valeurs non NULL.

LMD – fonctions statistiques

- Affichage - Exemples :

- Questions :
  - Quel est le prix maximal d'une voiture?
  - Quel est le prix maximal d'une Renault?

- Réponses :

```
SELECT MAX(prix) FROM VOITURE;
```

```
SELECT MAX(prix) FROM VOITURE
WHERE Marque='Renault';
```

- Affichages

Max(prix)
16000

Max(prix)
10000

LMD – fonctions statistiques

- Affichage - Exemples :

- Questions :
  - Quel est le nombre de Tuples de la table VOITURE?
  - Quel est le nombre de marque de voitures ?

- Réponses :

```
SELECT COUNT(*) FROM VOITURE;
```

```
SELECT COUNT(DISTINCT marque) FROM VOITURE;
```

- Affichages :

Count(*)	Count(distinct marque)
6	4

- Attention : sans distinct, les doublons seront considérés ➔

Count(marque)
6

LMD – fonctions statistiques

- Affichage - Exemples :

- Questions :
  - Quel est la somme des prix des voitures ayant une Puissance de 5 chevaux?
  - Quel est le prix moyen des voitures de 5 chevaux?

- Réponses :

```
SELECT SUM(Prix) FROM VOITURE WHERE Puissance=5;
```

```
SELECT AVG(Prix) FROM VOITURE  WHERE Puissance=5;
```

- Affichages :

SUM(Prix)	AVG(Prix)
23500	7833,33333

LMD - Affichage

- Affichage :

➢ GROUP BY : permet de partitionner la relation résultat selon les valeurs d'un ou de plusieurs attributs

– Les seuls noms de colonnes (en dehors des fonctions statistiques) qui peuvent apparaître dans le SELECT sont celles qui figurent dans le GROUP BY

➢ HAVING : définit les conditions que les groupes doivent respecter pour être retenus, elle sélectionne les partitions désirées

– Elle ne peut comprendre que des conditions dont le premier terme est une fonction statistique

– La clause HAVING est aux groupes (GROUP BY) ce que la clause WHERE est aux lignes (SELECT)

LMD - Affichage

- Affichage :

➢ ORDER BY : permet d'ordonner le résultat sur un ou plusieurs attributs, l'ordre pouvant être croissant (grâce au mot-clé ASC) ou décroissant (grâce au mot-clé DESC), ASC étant l'ordre par défaut

➢ Dans une requête SQL, la clause ORDER BY se situe juste après une clause WHERE ou après une clause GROUP BY

LMD – fonctions statistiques

- Fonctions statistiques - Exemples :

- Questions :
  - Quel est la somme des prix des voitures selon la Puissance dans la table VOITURE?

- Réponses :

```
SELECT SUM(Prix), Puissance FROM VOITURE
GROUP BY Puissance;
```

- Affichages :

SUM(Prix)	Puissance
23 500	5
13000	6
15250	7

LMD – fonctions statistiques

- Fonctions statistiques - Exemples :

- Questions :
  - Quel est la somme des prix des voitures selon la Puissance dans la table VOITURE (calculer le prix maximum sur ces partitions et limiter l'affichage aux partitions contenant au moins 2 éléments)?

- Réponses :

```
SELECT MAX(Prix), Puissance FROM VOITURE
GROUP BY Puissance HAVING COUNT(*) > 1 ;
```

- Affichages :

SUM(Prix)	Puissance
23 500	5
15250	7

LMD – fonctions statistiques

- Fonctions statistiques - Exemples :

- Questions :
  - Afficher l'ensemble des Tuples de la table VOITURE triés sur l'attribut Puissance selon un ordre croissant

- Réponses :

```
SELECT * FROM VOITURE ORDER BY Puissance ASC;
```

- Affichages :

Immatriculation	Marque	Couleur	Puissance	Prix
123ABC92	Peugeot	Noire	5	2 500
698KLM45	Citroën	Verte	5	5 000
956BZD59	Peugeot	Jaune	5	16 000
123ATT77	Rover	Bleue	6	13 000
123ABC75	Renault	Rouge	7	10 000
156AMN86	Renault	Bleue	7	5 250

LMD – fonctions statistiques

- Fonctions statistiques - Exemples :

- Questions :
  - Afficher l'ensemble des Tuples de la table VOITURE triés sur l'attribut Puissance selon un ordre décroissant puis sur l'attribut Couleur (ordre alphabétique) selon un ordre croissant

- Réponses :

```
SELECT * FROM VOITURE ORDER BY Puissance DESC, Couleur;
```

- Affichages :

Immatriculation	Marque	Couleur	Puissance	Prix
156AMN86	Renault	Bleue	7	5 250
123ABC75	Renault	Rouge	7	10 000
123ATT77	Rover	Bleue	6	13 000
956BZD59	Peugeot	Jaune	5	16 000
123ABC92	Peugeot	Noire	5	2 500
698KLM45	Citroën	Verte	5	5 000

LMD – opérateurs relationnels

- Opérateurs relationnels :

- Les principaux opérateurs sont:
  - union, intersection et différence

- Ces opérations sont applicables sur deux relations R1 et R2 qui ont le même schéma de table (nombre équivalent d'attributs et attributs identiques deux à deux)

LMD – opérateurs relationnels

- Opérateurs relationnels - Exemples :

- Questions :
  - Afficher l'ensemble des Tuples de la table VOITURE dont la Puissance est égale à 5 ou à 7 chevaux

- Réponses :

```
SELECT * FROM VOITURE WHERE Puissance='5'
UNION SELECT * FROM VOITURE WHERE Puissance='7';
```

⟺

```
SELECT * FROM VOITURE WHERE Puissance = 5 OR Puissance = 7 ;
```

- Affichages :

Immatriculation	Marque	Couleur	Puissance	Prix
123ABC75	Renault	Rouge	7	10 000
123ABC92	Peugeot	Noire	5	2 500
698KLM45	Citroën	Verte	5	5 000
956BZD59	Peugeot	Jaune	5	16 000
156AMN86	Renault	Bleue	7	5 250

LMD – Jointure

- La jointure :

➢ Les opérateurs de jointures donnent la possibilité d'appliquer des requêtes sur plusieurs tables d'une base de données

➢ La syntaxe est la suivante :

```
SELECT nom_champ FROM nom_table [As tab1] [INNER | {{LEFT | RIGHT |
FULL } [OUTER]}] JOIN  nom_table2 [As tab2] ON Condition;
```

```
SELECT nom_champ FROM nom_table [As tab1] [INNER | {{LEFT | RIGHT |
FULL } [OUTER]}] JOIN  nom_table2 [As tab2] USING (colonnes);
```

❑ La condition de la commande ON permet de comparer les tables jointes par l'intermédiaire de champs dont les valeurs sont identiques et comparables

❑ As : est optionnel, il permet d'utiliser un surnom de la table

LMD – Jointure

- La jointure :

➢ Les types de jointures possibles :

✓ INNER JOIN (par défaut) : toutes les paires correspondantes des lignes renvoyées et supprime les lignes n'ayant pas de correspondance entre les deux tables

✓ FULL OUTER JOIN : une ligne de la table de gauche ou de droite, ne respectant pas la condition de jointure, est comprise dans le jeu de résultats et que les colonnes de sortie correspondant à l'autre table comportent des valeurs NULL

✓ LEFT OUTER JOIN : toutes les lignes de la table de gauche ne respectant pas la condition de jointure sont incluses dans le jeu de résultats, et que les colonnes de sortie de l'autre table ont des valeurs NULL en plus de toutes les lignes renvoyées par la jointure interne

✓ RIGHT OUTER JOIN : toutes les lignes de la table de droite ne respectant pas la condition de jointure sont comprises dans le jeu de résultats, et que les colonnes de sortie correspondant à l'autre table ont des valeurs NULL en plus de toutes les lignes renvoyées par la jointure interne

LMD - Jointure

- Jointure - Exemples :

Voiture

Immatriculation	Marque	Couleur	Puissance	Prix
123ABC75	Renault	Rouge	7	10 000
123ABC92	Peugeot	Noire	5	2 500
123ATT77	Rover	Bleue	6	13 000
698KLM45	Citroën	Verte	5	5 000
956BZD59	Peugeot	Jaune	5	16 000
156AMN86	Renault	Bleue	7	5 250

Marques

Marque	siège
Renault	Paris
Peugeot	Lille

LMD – Jointure

- Jointure - Exemples :

- Questions :
  - Afficher l'immatriculation, la marque et la couleur d'une voiture qui existe dans la table Marque (jointure interne)

- Réponses :

```
SELECT immatriculation, Voiture.marque, couleur FROM Voiture WHERE
Voiture.marque=Marques.marque;
```

⟷

```
SELECT immatriculation, Voiture.marque, couleur FROM Voiture
INNER JOIN Marques ON Voiture.marque=Marques.marque;
```

- Affichages :

Immatriculation	Marque	couleur
123ABC75	Renault	Rouge
123ABC92	Peugeot	Noire
956BZD59	Peugeot	Jaune
156AMN86	Renault	Bleue

LMD – Jointure

- Jointure - Exemples :

- Questions :
  - Afficher les informations d'une voiture (jointure externe gauche avec la table Marque)

- Réponses :

```
SELECT * FROM Voiture LEFT OUTER JOIN Marques USING (marque);
```

- Affichages :

Immatriculation	Marque	Couleur	Puissance	Prix	Siège
123ABC92	Peugeot	Noire	5	2 500	Lille
698KLM45	Citroën	Verte	5	5 000	
956BZD59	Peugeot	Jaune	5	16 000	Lille
123ATT77	Rover	Bleue	6	13 000	
123ABC75	Renault	Rouge	7	10 000	Paris
156AMN86	Renault	Bleue	7	5 250	Paris

LMD - remarque

- Remarque :

➢ Il est possible d'imbriquer plusieurs requêtes. L'imbrication se fait dans la clause WHERE ou bien dans la clause FROM

- Exemples :

❑ Questions :

- Donner la requête qui permet d'afficher tous les Tuples de la table *VOITURE* dont le prix est supérieur à la moyenne des prix de tous les Tuples de cette même table

❑ Réponses :

```
SELECT * FROM VOITURE
WHERE Prix > (SELECT AVG(Prix) FROM VOITURE) ;
```

LMD – remarque

- Remarque :

- Une sous-interrogation (requête imbriquée) peut retourner un sous ensemble de zéro à n valeurs

- En fonction des conditions que l'on veut exprimer au niveau des valeurs renvoyées par la requête, on peut utiliser les prédicats IN, ANY, ALL ou EXISTS :

  - ✓ IN : l'opérateur de comparaison est l'égalité et l'opération logique entre les valeurs est OU
  - ✓ ANY : permet de vérifier si *au moins une valeur* de la liste satisfait la condition
  - ✓ ALL : permet de vérifier si la condition est réalisée pour *toutes les valeurs* de la liste
  - ✓ EXISTS : si la sous-interrogation renvoie un résultat, la valeur retournée est Vrai sinon la valeur Faux est retournée

LMD - remarque

- Exemple:

- Questions :
  - Produire les informations au sujet des CLIENTS qui ont passé au moins une commande
  - Commandes passées après la dernière Livraison (date ultérieure)

- Réponses :

```
SELECT * FROM CLIENTS WHERE EXISTS
(SELECT * FROM COMMANDE WHERE nClient = CLIENTS.nClient) ;
```

```
SELECT * FROM COMMANDE WHERE dateCommande > ALL
(SELECT dateLivraison FROM LIVRAISON) ;
```

LMD

- Plan :
  - Insertion
  - Affichage
  - Mise à jour
  - Suppression

LMD – Mise à jour

- Mise à jour (UPDATE):

➢ Il est possible de faire des mise à jour des données d'une BDD

➢ La syntaxe est la suivante :

```
UPDATE <nom_table> SET <attribut1> = <expression1>, <attribut2> = <expression2> ...
      [WHERE <condition>];
```

- Remarque : la clause WHERE est optionnelle, cependant sans le WHERE, la mise à jour affectera TOUS les Tuples de la relation.

LMD – Update

- Update - Exemples :

- Questions :
  - Donner la requête qui permet d'augmenter le Prix de vente de 10% de tous les Tuples de la table *VOITURE*
  - Donner la requête qui permet d'augmenter le Prix de vente de 10% des voiture de marque Renault

- Réponses :

```
UPDATE VOITURE SET Prix = Prix x 1.10;
```

```
UPDATE VOITURE SET Prix = Prix x 1.10 WHERE
Marque='Renault';
```

LMD

- Plan :
  - Insertion
  - Affichage
  - Mise à jour
  - Suppression

LMD – Suppression

- Suppression (DELETE):

➢ Il est possible de supprimer des Tuples ou une partie des Tuples d'une table

➢ La syntaxe est la suivante :

```
DELETE FROM <nom_table> [WHERE <condition>];
```

- Remarque : la clause WHERE est optionnelle, cependant sans le WHERE, la suppression affectera TOUS les Tuples de la relation.

LMD – Suppression

- DELETE - Exemples :

- Questions :
  - Donner la requête qui supprime tous les Tuples de la table *VOITURE*
  - Donner la requête qui supprime les voitures de marque Renault

- Réponses :

```
DELETE FROM VOITURE;
```

```
DELETE FROM VOITURE
WHERE Marque='Renault';
```

Langage de contrôle des données

- LCD (langage de contrôle des données) : permet de gérer les privilèges, c'est-à-dire les utilisateurs et les actions qu'ils peuvent entreprendre

- Les instructions du LCD sont : GRANT, REVOKE

- Plan :
  - GRANT : permet d'attribuer un privilège à différents utilisateurs sur différents objets
  - REVOKE : permet de révoquer, c'est-à-dire "retirer" un privilège

LCD - GRANT

- LCD – GRANT:

➢ La syntaxe est la suivante :

```
GRANT liste_des_privilèges | ALL
PRIVILEGES ON objets
TO nom_autorisé | PUBLIC
[WITH GRANT OPTION];
```

Liste des privilèges

```
SELECT |
DELETE |
INSERT [listeColonnes]|
UPDATE [listeColonnes]|
REFERENCES listeColonnes|
USAGE
```

objets

```
[TABLE] nomTable |
DOMAIN nomDomaine |
CHARACTER SET nomCharacterSet
COLLATION nomCollation
TRANSLATION nomTranslation
```

LCD – GRANT

- GRANT - Exemples :

- Requête qui donne l'ensemble des droits d'accès aux utilisateurs *user2 et user3* uniquement sur les colonnes *Immatriculation* et *Prix* de la table *VOITURE*

```
GRANT ALL PRIVILEGES (Immatriculation, Prix) ON VOITURE TO user2, user3;
```

- Requête qui donne uniquement le droit d'insertion de Tuples aux utilisateurs sur la table *VOITURE*

```
GRANT INSERT ON PERSONNE TO PUBLIC;
```

- La requête suivante permet de donner à *user2* tous les droits sur la table VOITURE, il lui accorde également l'autorisation de transférer ces privilèges

```
GRANT ALL PRIVILEGES ON VOITURE TO user2 WITH GRANT OPTION;
```

LCD - REVOKE

- LCD – REVOKE:

➢ La syntaxe est la suivante :

```
REVOKE ALL PRIVILEGES | liste_des_privilèges | GRANT OPTION FOR
listepriv
ON objets
FROM nom_utilisateur | PUBLIC
[ RESTRICT | CASCADE ];
```

RESTRICT

Si le ou les utilisateurs visés ont cédés leurs droits à d'autres, un message d'erreur apparaîtra et le SGBDR refusera de révoquer le ou les droits

CASCADE

Entraînera la révocation des droits cédés à la manière d'un château de carte

Langage de contrôle des transactions

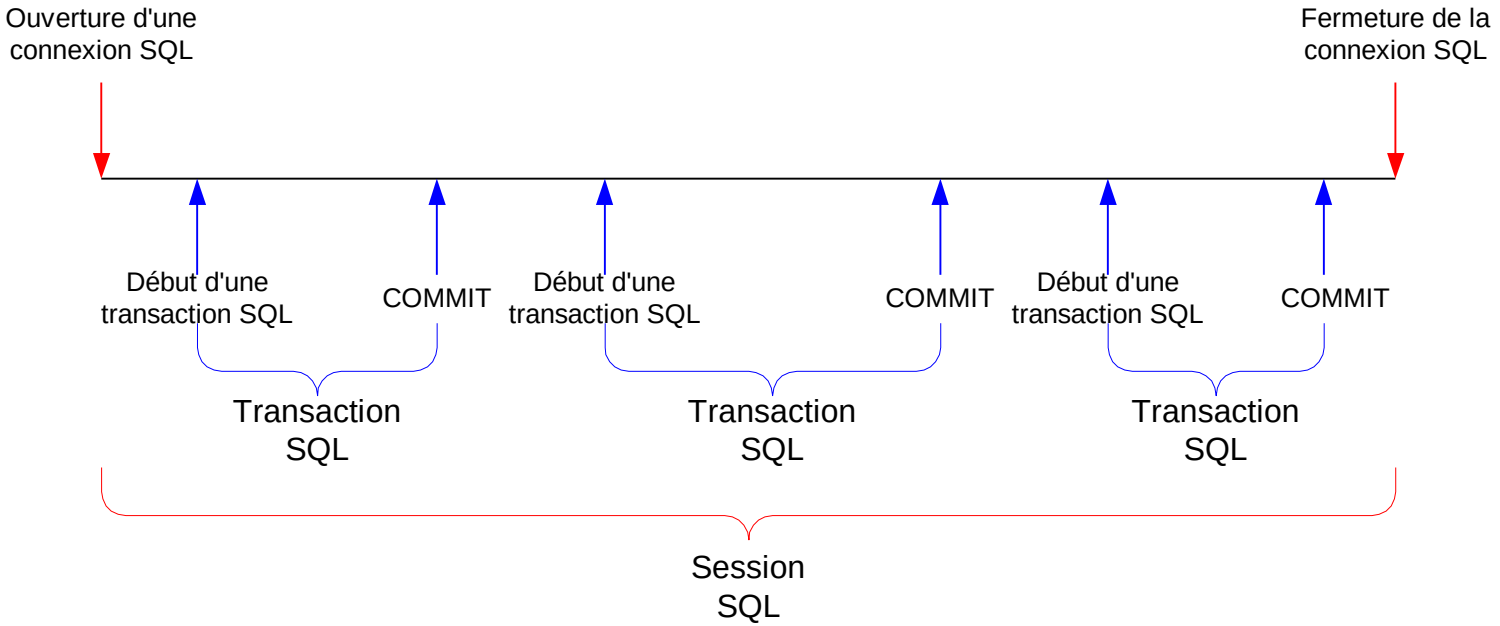
- LCT (langage de contrôle des transactions) : permet de gérer les transactions, c'est-à-dire rendre atomique divers ordres enchaînés en séquence

- Les instructions du LCT sont : COMMIT, ROLLBACK

- Plan :
  - Propriétés ACID
  - COMMIT : permet à l'utilisateur de fixer le moment où les modifications en cours affecteront la base de données
  - ROLLBACK : permet à l'utilisateur de ne pas valider les dernières modifications en cours dans la base de données

LCT

- LCT (langage de contrôle des transactions) :



- Début de transaction (implicite en SQL)
- Fin de transaction
  - Confirmer (COMMIT WORK en SQL)
  - Annuler (ROLLBACK WORK en SQL)

LCT - ACID

- ACID :

- *A*tomicité : une transaction s'effectue ou pas (tout ou rien), une transaction ne peut être partiellement effectuée

- *C*ohérence : le résultat ou les changements induits par une transaction doivent impérativement préserver la cohérence de la base de données

- *I*solation : les transactions sont isolées les unes des autres. Les effets de la transaction ne sont pas perceptibles tant que celle-ci n'est pas terminée

- *D*urabilité : une fois validée, une transaction doit perdurer, c'est-à-dire que les données sont persistantes même s'il s'ensuit une défaillance dans le système

LCT - ACID

- COMMIT :

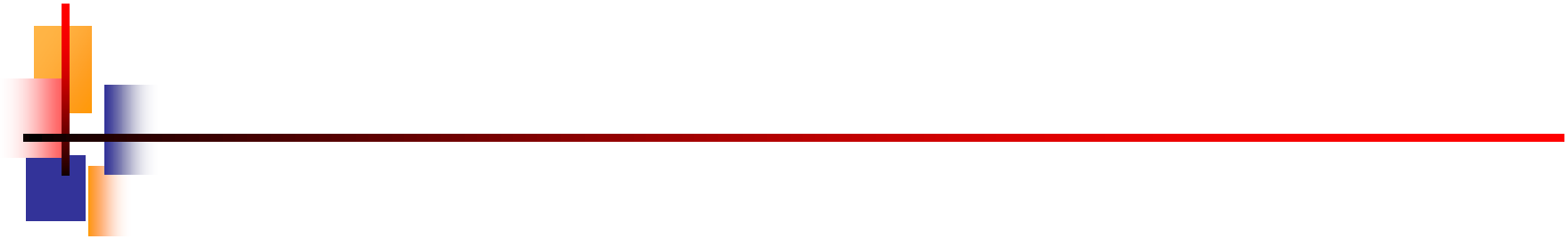
- La commande COMMIT permet à l'utilisateur de terminer et de valider (écrit) la transaction
- Une transaction (ou une partie de transaction) qui n'a pas encore été enregistrée définitivement n'est visible que pour l'utilisateur qui l'introduit
- Avant l'exécution de l'instruction COMMIT, il est possible de restaurer la base par ROLLBACK, c'est-à-dire d'éliminer les modifications récentes
- Après l'enregistrement définitif d'une transaction par COMMIT, il n'est plus possible de restaurer l'état antérieur par ROLLBACK
- S'il apparaît après coup qu'une transaction doit être modifiée ou corrigée, nous ne pourrons pas effectuer cette modification qu'au moyen d'une autre instruction SQL comme UPDATE ou DELETE.

LCT - ACID

- ROLLBACK :

  - La commande ROLLBACK permet à l'utilisateur de terminer et d'annuler la transaction

  - Dans le cas d'une défaillance du système, l'intégrité de la base peut être préservée par une option ROLLBACK automatique qui élimine les transactions inachevées et empêche donc qu'elles soient introduites dans la base



Partie 5 : Divers

- JDBC
- PostgreSQL

JDBC - Introduction

- Besoin d'accès distant aux bases de données
  - depuis un langage standard (C, C++, Java, …)
  - ou même un outil bureautique (Excel, Word, …)

✓ Microsoft avec ODBC permet l'accessibilité depuis Windows à toute les bases de données du marché

✓ L'objectif est de permettre de dialoguer avec une base de données distante en utilisant les instructions SQL (en forme de chaînes de caractères) à partir du langage hôte, et de présenter les ensembles de résultats dans une manière compréhensible pour le langage hôte.

JDBC - Introduction

- Besoin d'accès distant aux bases de données

➢ Deux approches d'intégration existent:
  - Instructions SQL imbriquées dans le langage hôte (« Embedded SQL », SQLJ)
  - Création d'une API spéciale pour appeler les commandes SQL (JDBC)

- Les relations SQL sont des (multi)ensembles de Tuples qui n'imposent aucune limite a priori sur le nombre de Tuples :
  - Les langages de programmation traditionnels (C, C++, etc.) n'ont pas une telle structure de données
  - SQL supporte un mécanisme appelé *curseur* pour résoudre ce problème.

JDBC - Introduction

- API JDBC (*Java DataBase Connectivity)* :

✓ Est une API Java (ensemble de classes) permettant de développer des applications capables de se connecter et d'accéder à des bases de données distantes

✓ Les API ont été développées de telle façon à permettre à un programme de se connecter à n'importe quelle base de données en utilisant la même syntaxe (indépendante du SGBD)

- JDBC :

- API standard pour JAVA
- http://java.sun.com/products/jdbc/
- Ensemble de classes
- Besoin d 'installer un pilote JDBC dans l'environnement JAVA.

JDBC

- API JDBC (*Java DataBase Connectivity)* :

✓ Étapes pour soumettre une requête:

1. Choisir le driver JDBC
2. Connecter à la source de données
3. Exécuter les instructions SQL

JDBC - Driver

- JDBC – pilotes :

✓ Tous les drivers sont gérés par la classe DriverManager

✓ Pour choix un driver JDBC, 3 voies sont possibles:

– Dans le code Java:

Class.forName("oracle/jdbc.driver.Oracledriver");

– En démarrant une application Java (à la ligne des commandes):
-Djdbc.drivers=oracle/jdbc.driver

– En créant un objet driver explicitement

JDBC - Connexion

- JDBC – connexion :

✓ L'interaction avec une source de données se fait par l'entremise d'une session qui est démarrée par la création d'un objet de type ''Connection''

✓ Chaque session est démarrée via un URL JDBC

✓ URL JDBC (i.e. un URL utilisant le protocole JDBC):

jdbc:<sousprotocole>:<autresParametres>

JDBC - Connexion

- JDBC – connexion :

- Exemple :

  Établir une connexion à une BD Oracle précise

```
String url="jdbc:oracle:www.bookstore.com:3083";
connexion con1;

try{
        con1 = DriverManager.getconnexion(url,usedId,password);
}catch (SQLException e) {
 …}
```

JDBC - SQL

- JDBC – Connexion (les interfaces pour la connexion) :

✓ Détermine le degré d'isolation de la connexion en cours :

public int getTransactionIsolation() et void setTransactionIsolation(int level)

✓ Spécifie si les transactions dans la connexion sont pour lecture seulement :

public Boolean getReadOnly() et void setReadOnly(boolean b)

✓ Vérifie si la connexion est encore ouverte :

public boolean isClosed()

✓ Contrôle les transactions :

Si autocommit est vrai, chaque instruction SQL est considérée comme une transaction autonome. Sinon, la transaction est validée par commit(), ou abandonnée par rollback().

public boolean getAutoCommit() et void setAutoCommit(boolean b)

JDBC - SQL

- JDBC – SQL :

✓ Il y a trois manière différentes pour exécuter des requêtes SQL :
  - Statement (permet des requêtes avec soit des instructions SQL statiques soit dynamiques)
  - PreparedStatement (instructions semi-statiques)
  - CallableStatement (procédures stockées)

✓ La classe PreparedStatement génère dynamiquement des instructions SQL précompilées et paramétrisées:
  - Structure fixe
  - Valeurs des paramètres déterminées pendant l'exécution

JDBC - SQL

- JDBC – SQL :

✓ Exemple de requêtes :

```
String sql="INSERT INTO Sailors VALUES(?,?,?,?)";
PreparedStatment pstmt=con.prepareStatement(sql);

pstmt.clearParameters();
//insertion des paramètres : p1, p2, p3 et p4
pstmt.setInt(1,p1);
pstmt.setString(2,p2);
pstmt.setInt(3, p3);
pstmt.setFloat(4,p4);

// Nous savons que aucune ligne n'est retournées (INSERT)
// d'où nous utilisons executeUpdate()
int numRows = pstmt.executeUpdate();
```

JDBC - SQL

- SQL - résultats :

✓ PreparedStatement.executeUpdate : retourne seulement le nombre de ligne affectées

✓ PreparedStatement.executeQuery : retourne les données, repris dans un objet ResultSet

```
ResultSet rs=pstmt.executeQuery(sql);

While (rs.next()) {
  // traiter les données
}
```

JDBC - SQL

- SQL - résultats :

✓ PreparedStatement.executeUpdate : retourne seulement le nombre de ligne affectées

✓ PreparedStatement.executeQuery : retourne les données, repris dans un objet ResultSet

- Un ResultSet propose plusieurs fonctionnalités
- previous(): reculer d'une ligne
- absolute(int num): avancer à la ligne dont le numéro d'ordre est indiqué
- relative (int num): avancer ou reculer selon la valeur indiquée
- first() et last()

```
ResultSet rs=pstmt.executeQuery(sql);

While (rs.next()) {
  // traiter les données
}
```

JDBC - SQL

- JDBC – SQL :

✓ Correspondance entre les Types de Données: Java vs SQL

SQL Type	Java class	ResultSet get method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

JDBC

JDBC – exemple complet :

```
public static void main(String args[]){
        String url = "adresse url du serveur de la BDD";
        Connection con=null;
        Statement stmt;
        String query = "select nom,couleur,poids from produits;";
        try {
                Class.forName("org.postgresql.Driver");
                con = DriverManager.getConnection(url,"login", "motpass");
                stmt = con.createStatement();
                ResultSet rs = stmt.executeQuery(query);
                System.out.println("Liste des produits:");
                while (rs.next()){
                        String n = rs.getString(1); // nom
                        String c = rs.getString(2); // couleur
                        float p = rs.getFloat(3); // poids
                        System.out.println(p + " " + n + " " + p);
                }
        }catch(SQLException ee){
                ee.printStackTrace();
        }catch (Exception e){
                e.printStackTrace();
        }
        if (con!=null) try {con.close();} catch(Exception e){} //
}
```

PostgreSQL

- PostgreSQL :

➢ Il y a plusieurs manières de mesurer un logiciel : les fonctionnalités, les performances, la fiabilité, le support et le prix

✓ Fonctionnalités

– PostgreSQL possède la plupart des fonctionnalités présentes dans les SGBD commerciaux, comme :

  Les transactions, les requêtes imbriquées, les déclencheurs, les vues, l'intégrité référentielle par clés étrangères, et le verrouillage sophistiqué

– En plus, il propose d'autre fonctionnalités : comme les types définis par l'utilisateur, l'héritage, les règles, et le contrôle de concurrence par multi-versionnage pour réduire les contentions de verrouillage

PostgreSQL

- PostgreSQL :

✓ Performances

- PostgreSQL a des performances similaires aux autres bases de données commerciales et open source (il est plus rapide pour certaines opérations, plus lent pour d'autres)

- Par rapport à MySQL ou d'autres SGBD plus léger, PostgreSQL est plus rapide pour de nombreux utilisateurs, des requêtes complexes et une charge pour les requêtes de lecture/écriture

- MySQL est plus rapide pour des requêtes SELECT simples effectuées par quelques utilisateurs. Bien sûr, MySQL ne possède aucune des fonctionnalités de la section *Fonctionnalités*

- MySQL est une société qui distribue son produit via l'open source et requiert une licence commerciale pour les logiciels propriétaires, donc pas une communauté de développement open source comme PostgreSQL

PostgreSQL

- PostgreSQL :

✓ Fiabilité

- Il est évident qu'un SGBD doit être fiable
- PostgreSQL offre des versions bien testées, du code stable ne contenant qu'un minimum de bogues
- Chaque version a au moins un mois de tests, avec un historique de versions fournissant des versions stables et robustes, prêtes pour une utilisation en environnement de production

PostgreSQL

- PostgreSQL :

✓ Support

- Listes de diffusion offrent un contact avec un large groupe de développeurs et d'utilisateurs afin d'aider a la résolution des problèmes rencontrés
- PostgreSQL ne peut pas toujours garantir un correctif mais les SGBD commerciaux ne le garantissent pas toujours non plus
- L'accès direct aux développeurs, à la communauté d'utilisateurs, aux manuels, et au code source, fait du support pour PostgreSQL un support supérieur aux autres SGBD

✓ Prix

- Gratuité pour tous les usages, commerciaux et non commerciaux

Références

PostgreSQL - Documentations en lignes :

- Quelques liens utiles:
  - http://www.postgresql.org/docs/8.3/interactive/index.html
  - http://postgresql.developpez.com/faq/

DB Designer 4 :

- Quelques liens utiles:
  - http://www.fabforce.net/dbdesigner4/docs.php

JDBC :

- Quelques liens utiles:
  - http://java.sun.com/javase/technologies/database/