Les Systèmes de Gestion de Bases de Données (SGBD)

Quelques références

- S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley
- J.C. Date, A Guide to the SQL Standard, Addison-Wesley
- J.C. Date, A Guide to DB2, Addison-Wesley
- R. Elmasri, S. Navathe, Conception et architecture des bases de données, 4ème ed., publié par Pearson Education.
- H. Garcia-Molina, J. Ullman and J. Widom, Implementation of Database Systems, Prentice Hall, 1999.
- G. GARDARIN, Bases de Données, Eyrolles, 6ème tirage, 2005.
- R. Ramakrishnan et J. Gehrke DATABASE MANAGEMENT SYSTEMS, MacGraw Hill
- M. SCHOLL, B. AMANN, P. RIGAUX, V. CHRISTOPHIDES, D. VODISLAV, Polycopié de Bases de Données, librairie des Arts et Métiers.
- Ullman J.D. and Widom J. A First Course in Database Systems, Prentice Hall, 1997
- Ullman J.D. Principles of Database and Knowledge-Base Systems, 2 volumes, Computer Science Press

Historique

- De tout temps, toute organisation (société) a cherché à
- structurer ses informations.
- Avant l'avènement de l'informatique : informations sous formes de fiches, classées par ordre alphabétique, chronologique, …,
- Supports informatiques ==> infos mémorisées sur supports magnétiques, …

- ==> les organiser pour pouvoir travailler dessus (retrouver telle info, voir sa valeur, supprimer, mettre à jour, …) => **notion de fichiers**.

Historique (2)

- <= année 60 : organisation classique en fichiers, gérés par des SGF
- Années 60 : 1ère génération de SGBD : le niveau conceptuel est très lié à la représentation des données sur les supports physiques
- ==> modèle hiérarchique, modèle réseau
- 1970-1980 : 2ère génération : modèle plus indépendant des supports : ==> modèle relationnel
- débuts des années 80 : 3ère génération :

  ==> modèle à objets, objet-relationnel, ...

Fichiers et SGF

Fichier : ensemble structuré d ’informations relatives à une entité (objet) donnée => supports : papier, disque, disquettes, CD,…

En informatique : support informatique (disque, CD, …)

Exemple : fichier EMPLOYES(nom, prenom, adr, categorie, num-ss, ….)

réalisation de ce fichier :

Dupont jean Paris 1601167576769 A ….

Durand Marc Toulon 1551267574556 B ….

Un fichier = ensemble d ’articles (enregistrements).

Un article = ensemble de rubriques (attributs)

Fichiers et SGF (2)

Organisation et accès aux fichiers :

1. Organisation séquentielle :

Les articles sont triés par ordre croiss. ou décr. du num. d ’accès.

L ’accès se fait en séquentiel : i^{ème} article accédé après avoir parcouru les (i-1) articles précédents.

=> utilisé pour les infos auxquelles on n ’accède pas souvent.

=> le support peut être à accès direct (disque,…) ou séquentiel (bande,…)

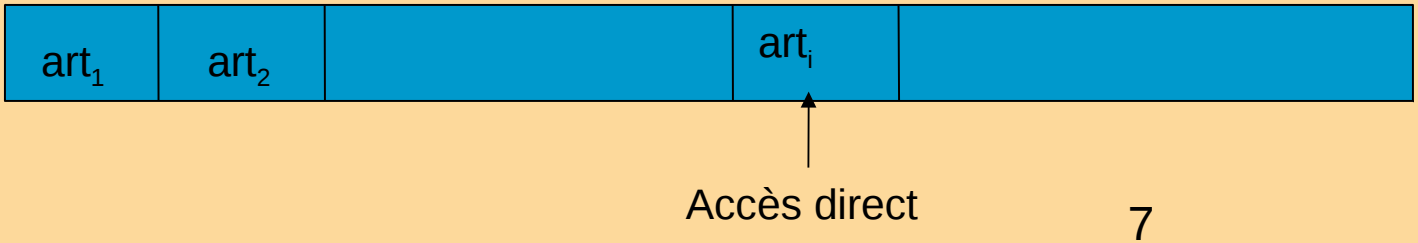
Fichiers et SGF (3)

2. Organisation à accès direct : les articles ne sont pas nécessairement ordonnées. Utilisée pour les infos auxquelles on accède aléatoirement.

L ’accès à un article se fait directement à partir de sa clé d ’identification (unique). Il y a toujours une relation entre la clé et l ’adresse physique(exple : clé = adresse, clé = adresse + cste, …).

==> bien adapté aux systèmes transactionnels (interrogation, màj fréquentes de quelques articles).

Supports : disques, CD-ROM, … tout support permettant l ’accès direct.

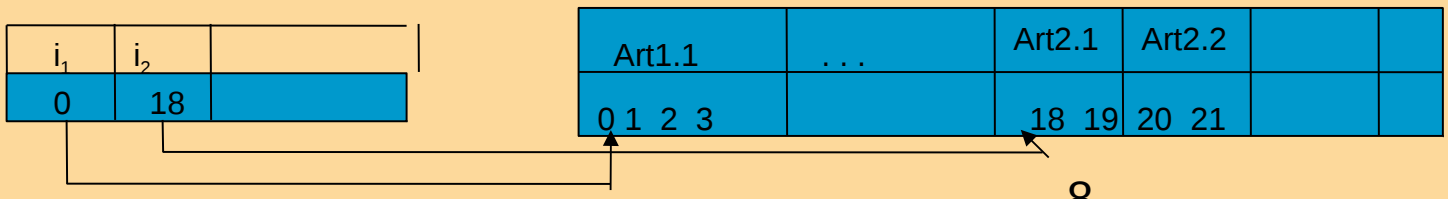


Fichiers et SGF (4)

3. Fichiers organisés en séquentiel indexé : compromis entre séquentiel et direct.

Accès en 2 temps :

- direct sur un élément d ’une table (l ’index)
- cette élément est un pointeur sur une zone du fichier, qui est ensuite parcourue séquentiellement pour trouver l ’élément cherché



Fichiers et SGF (5)

SGF = logiciel (ou ens. de log). Qui permet de gérer les fichiers application par application. Exmple : un syst. qui gère un fichier paie des employés, un fichier de gestion de pièces de rechange, gestion des congés, des absences, …

Inconvénients :

une application --> un programme,

absence de conception globale =>

-non exhaustivité : risque d ’infos manquantes,

-redondance : mêmes infos dans plusieurs fichiers

=> perte de place, risque d ’incohérence de màj,

-associations entre fichiers non exploitées

==> apparition du concept de **base de données**.

Base de données et SGBD

- Base de données = une ensemble structuré de données accessibles par l ’ordinateur pour satisfaire simultanément plusieurs utilisateurs
- SGBD = logiciel(s) qui permet d ’interagir avec une BD.

- Particularités de l ’organisation Bases de Données:
  - description et mémorisation des données
  - coûts réduits de saisie, stockage et màj,
  - partageabilité
  - contrôle
  - disponibilité, exactitude, cohérence et protection des données
  - intégrité, sécurité, confidentialité
  - conception a priori

Objectifs assignés aux SGBD

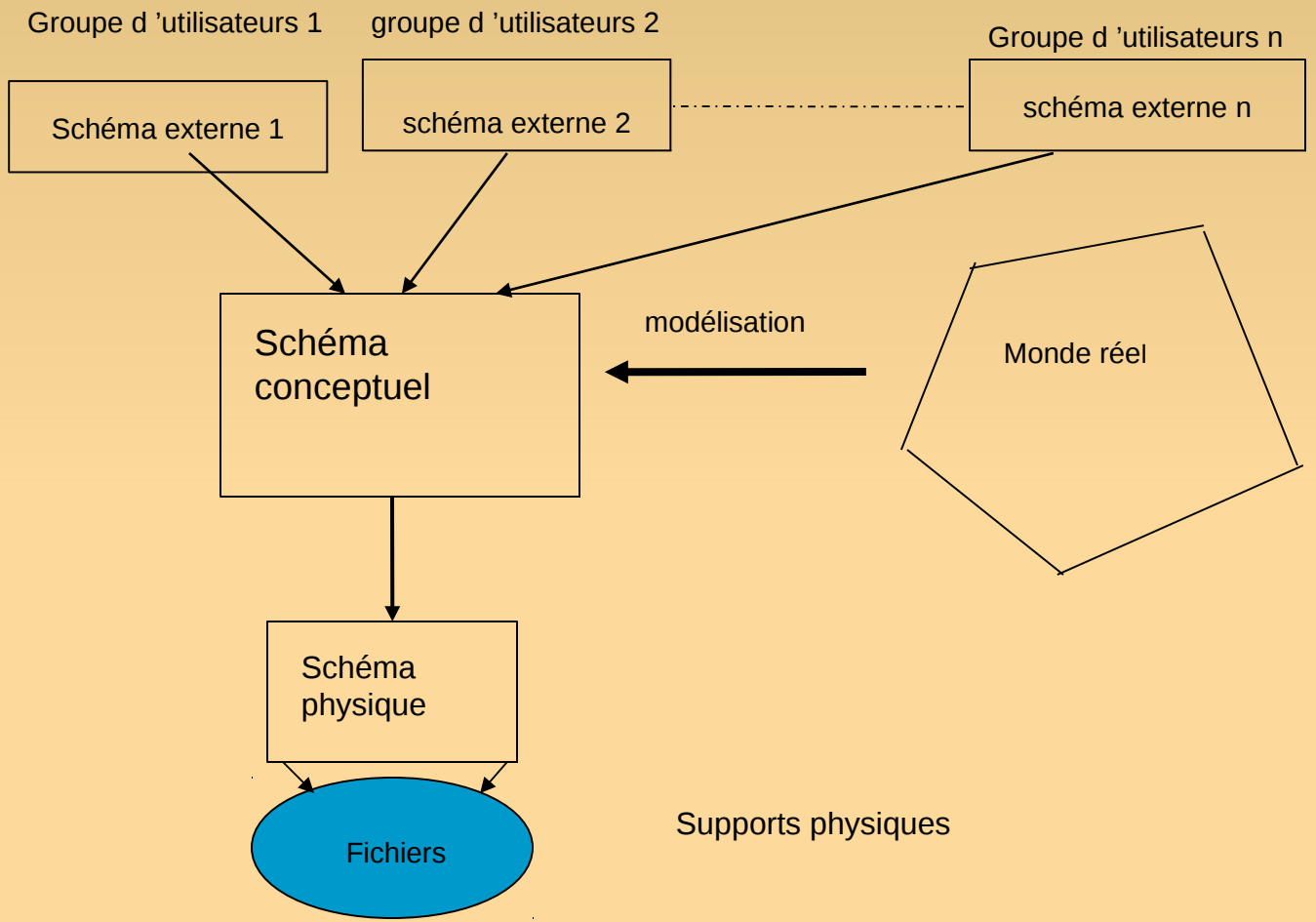
- Assurer :
  - **l ’indépendance physique** : une modification dans l ’organisation des données sur les supports physiques ne doit pas influer sur les programmes d ’applications
  - **l ’indépendance logique** : une modification dans l ’organisation logique des données (ex. ajout d ’une rubrique à un fichier) ne doit pas influer sur les programmes d ’applications non concernés.
- Manipulation aisée des données : un non-informaticien doit pouvoir manipuler facilement la base (interrogation et màj)
- Administration facile des BD : outils pour décrire, permettre d ’effectuer le suivi des structures, autoriser l ’évolution, contrôler le fonctionnement (tâche du DBA-DataBase Administrator)

Objectifs assignés aux SGBD (2)

- Efficacité **d ’accès** aux données : garantir un bon débit (trans. par seconde) et un bon temps de réponse (temps moyen de réponse des transactions)
- **Redondance contrôlée** des données : optimiser le volume de stockage, pas de màj multiples, pas ou peu de redondances
- Maintien de la **cohérence** des données, (ex. âge > 0), et le SGBD doit veiller à ce que les applications respectent cette règle (qui est une contrainte d ’intégrité)
- **Partage** des données entre plusieurs utilisateurs et applications en assurant la sécurité d ’accès en cas de conflit.
- **Sécurité** : données protégées contre les accès non autorisés et, en cas d ’incidents, restaurer un état cohérent de la base.

- ===> D'où les fonctions d ’un SGBD :
  - description des données (via le LDD)
  - recherche, mises à jour et transformations (via le LMD)
  - contrôle de l ’intégrité des données (respect des CI)
  - gestion des transactions (atomicité, reprise après panne, …) et sécurité (mots de passe, …)

Niveaux de représentation d ’une BD (rapport de l ’ANSI/SPARC 75)



- Dans la pratique, 4 niveaux :
  - niveau externe (sous-schéma conceptuels)
  - niveau conceptuel (en général, le modèle entité/association)
  - niveau logique (modèle hiérarchique, réseau, **relationnel**, objet)
  - niveau physique (fichiers, index, …)

- Niveau interne : comment sont stockées les données sur les supports physiques

  --> géré par le SGBD

- Niveau conceptuel : passage du monde réel au monde conceptuel via un modèle (exemple : le modèle entité/association). Doit être indépendant de toute implantation (de toute machine)

  --> géré par le concepteur de la BD

- Niveau logique : passage du modèle conceptuel à un modèle de bases de données (relationnel, objet, …), en vue de l ’implantation sur machine

  --> géré par le concepteur de la BD

- Niveau externe : création de vues (parties de la base de données) sur lesquelles des groupes d ’utilisateurs ont le droit de travailler (interroger, insérer, modifier et/ou supprimer -> selon les autorisations)

  --> géré par le concepteur de la BD

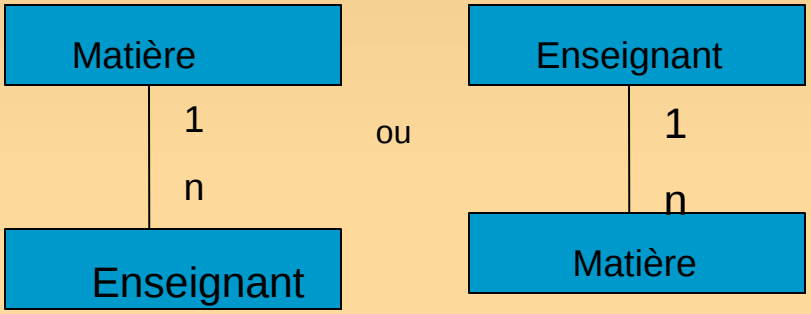
  --> le travail sur les vues est effectué par les utilisateurs.

Cycle de vie d ’une base de données

- Concevoir (analyse **=> modèle (E/A par exemple)** : *concepteur (ou administrateur)*
- Créer la structure **(modèle logique (ex. relationnel)** -> SGBD particulier): *idem*
- Implanter la base de données **(LDD+insertions initiales)** : *administrateur*
- Optimiser la base de données (index, tuning, …) : *administrateur*
- Manipuler (LMD: insérer, màj, supprimer) : *utilisateur*
- Maintenir (requêtes spécifiques) : administrateur

Le modèle hiérarchique

- généralise les notions du langage COBOL
- liens étroits entre données et traitements
- visualisation des données sous forme arborescente (hiérarchie) => nœuds = les classes d ’objets, arcs = associations. Un seul nœud racine => liens 1 à plusieurs uniquement.
- Exemple : Enseignants/Matières
- Schéma conceptuel :



2 classes d ’entités avec leur description

ENSEIGNANT {code-e, nom-e, prenom-e, categ-e, MATIERE { code-m, lib-m, coef-m, heures-m}}

E1	Talin Michel	Maître de conférences	
M1	Info	6	9
M2	Gestion	3	1
E2	Ans Jean-C.	Certifié	
M3	Maths	6	13
E3	Decaux F.	Agrégée	
M2	Gestion	3	5
M4	Compta	3	4
M5	Maths Fi.	1	2
E4	Carick J.M.	Certifié	
M3	Maths	6	6
M1	Info	6	7
E5	Bage R.	Vacataire	

Commentaires

- 5 occurrences hiérarchiques (une par enseignant)
- à chaque enseignant ==> une ou plusieurs matières (implantation du schéma 2 précédent)
- l ’accès aux données s ’effectue toujours à partir du supérieur hiérarchique (ici l ’enseignant)

- **Exemple de manipulation :**
- ***Q1. Trouver les matières enseignées par l ’enseignant de code E3***
- ***Début***
  - *lire (enseignant)*
  - *tantque (code-e <> « E3 » et (non fin-de-fichier)*
  - *faire lire (enseignant)*
  - *fintantque*
  - *Si code-e = « E3 »*
  - *alors lire (matière)*
    - *tantque matière existe*
    - *faire écrire(lib-m), lire (matière)*
    - *fintantque*
  - *Sinon écrire(l ’enseignant n ’existe pas) finsi*
- *Fin*

Q2. Trouver les noms des enseignants de la matière de code M1

- *Début*
  - *tantque (non fin-de-fichier)*
  - *faire lire (enseignant)*
    - *Lire (matière)*
    - *fairtantque matière existe*
    - *faire Si code-m = « M1 »*
      - *Alors écrire(nom-e, prenom-e)*
      - *Finsi*
      - *lire (matière)*
    - *fintantque*
  - *fintantque*

- *Fin*

Commentaires

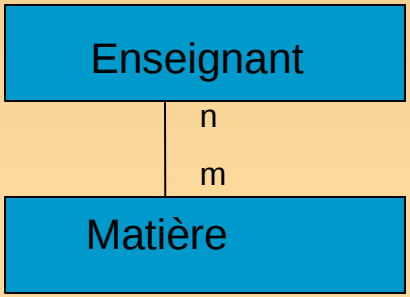
- Liste de toutes les matières => analyser, pour chaque enseignant, les matières qu ’il enseigne
- contrainte hiérarchique très forte : nécessité de connaître les chemins d ’accès

  - CONTRAINTES DU MODELE HIERARCHIQUE

- On doit spécifier COMMENT accéder aux données
- Anomalies d ’ajout, suppression et mise à jour. Exemple :
  - si ajout d ’une nouvelle matière => ajout d ’un enseignant (fictif !)
  - si suppression d ’un enseignant d ’une matière (qu ’il est seul à enseigner) => les infos sur cette matière disparaissent
  - si on veut modifier le coeff. D ’une matière ==> parcourir toute la base
- Le modèle hiérarchique ==> modélisation des liens (1:n) uniquement (liens hiérarchiques.

Le Modèle réseau

- Amélioration du modèle hiérarchique : représente les liens (n:m)
- exemple : un enseignant enseigne plusieurs matière et
  - une matière est enseignée par plusieurs enseignants
- schéma conceptuel :



•Création d ’un lien (SET), si association du type (1:1) ou (1:n) (exemple : 1 enseignant enseigne 1 seule matière (1:1) , ou 1 enseignant enseigne plusieurs matières (1:n)).

•Si association (n:m), utilisation d ’un artifice : la création d ’une nouvelle entité (RECORD). On construit alors 2 liens 1:n entre cette nouvelle entité et les entité de base.

•Exemple : Gestion des enseignants et matières. On a 2 entités de base (RECORD) : Enseignant et Matière. On crée un nouveau RECORD (que l ’on appellera E-M), qui permettra de réaliser la liaison entre Enseignant et Matière.

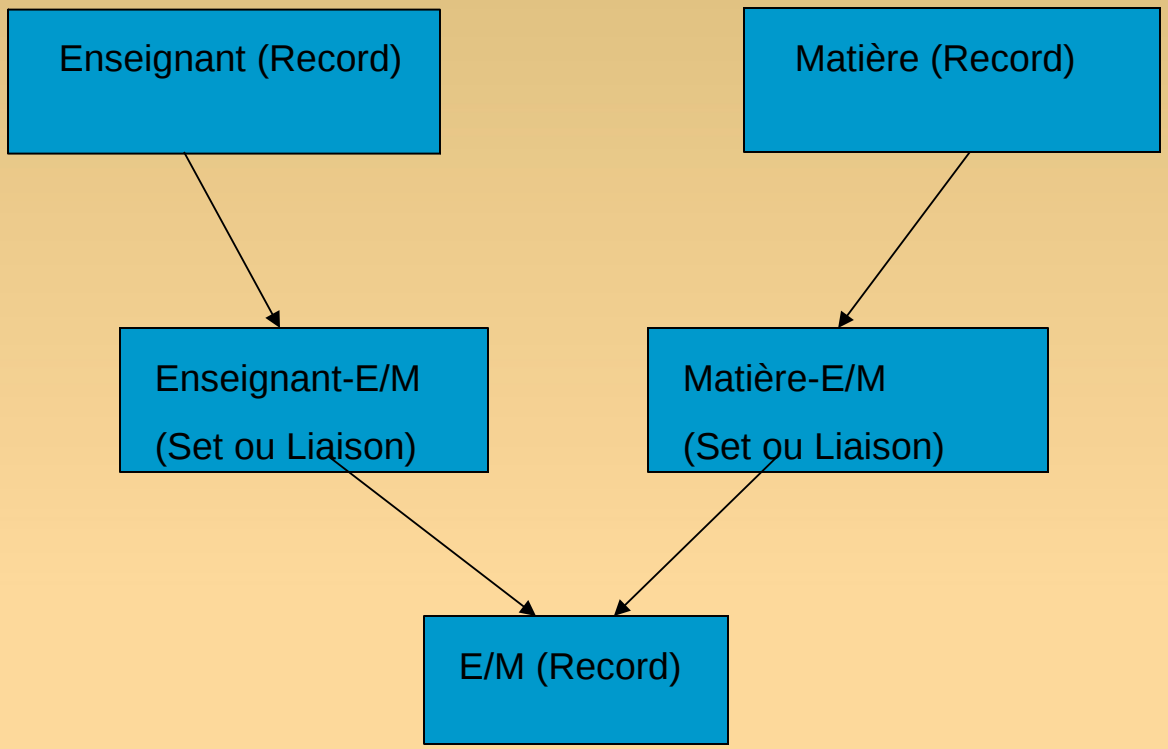


Fig. Modèle réseau (Base de donnée scolarité)

L ’association (n:m) est décrite à l ’aide de 2 liaisons (set) et est mise en œuvre avec des pointeurs

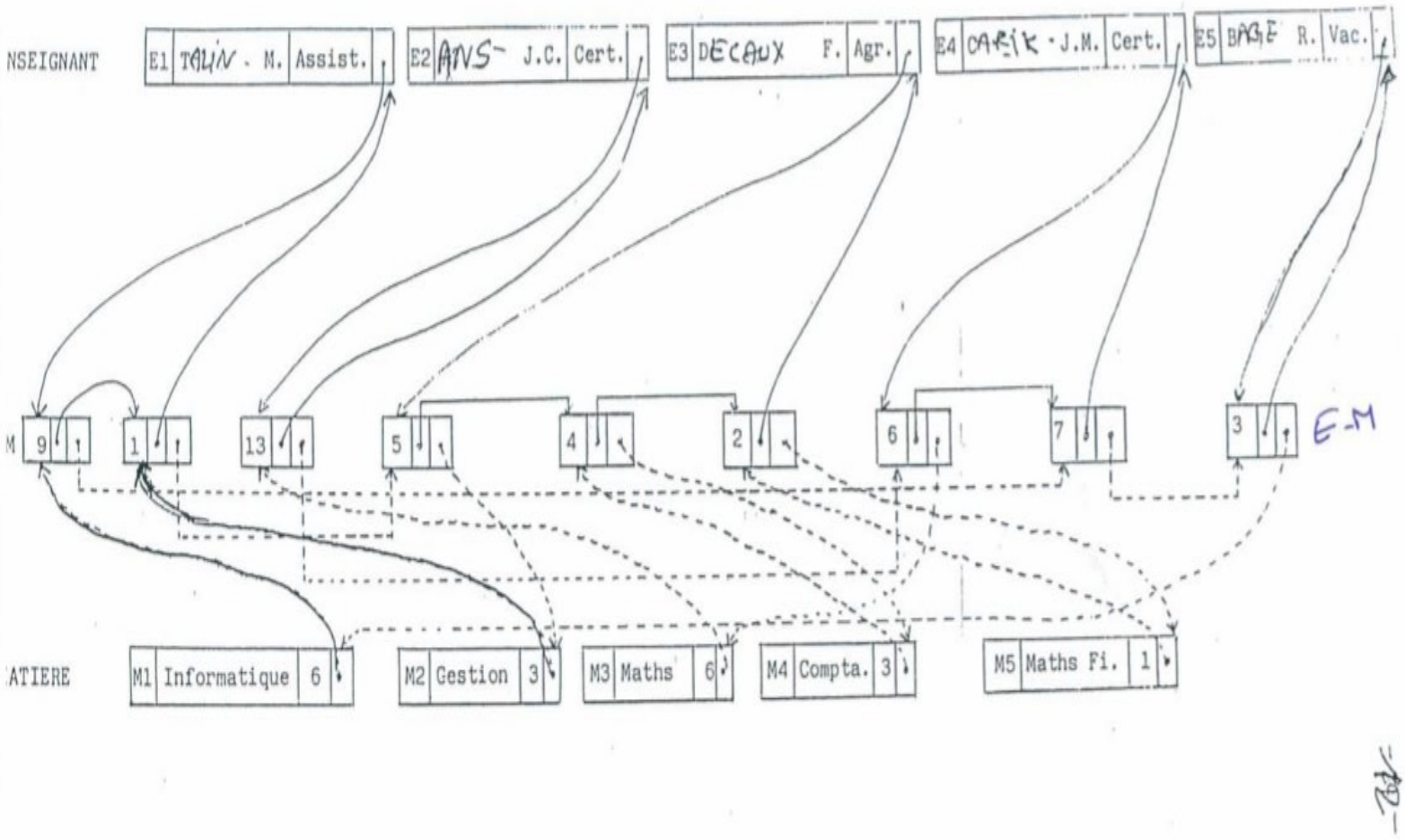


Figure 5 : Structure réseau

Commentaires sur le modèle réseau
(cf. figure sur l ’implantation du modèle réseau -chaînages-)

- Le LMD d ’un SGBD réseau permet le parcours des liens qui lient les Records
- Les 2 interrogations précédentes (base de données scolarité) peuvent être réalisées indifféremment à partir de Enseignant ou de Matière, mais la stratégie diffère. Par exemple, les temps de réponse seront différents si on cherche le nombre d ’heures effectuées par l ’enseignant E3 dans la matière M2, selon que l ’on commence la recherche par l ’enseignant ou par la matière.
- Les opérations classiques sont simplifiées par rapport au modèle hiérarchique.
  - - adjonction : on peut ajouter une nouvelle matière sans obligatoirement lui affecter un enseignant
  - la suppression : on peut supprimer un enseignant sans supprimer les matières qu’il enseigne
  - modification : pas de problème particulier

- Inconvénients du modèle réseau

- parcourt des chaînages (chemins d ’accès)
- gestion complexe des pointeurs

Le modèle relationnel (Brève présentation)

- Conçu par E.F Codd (1970)
- basé sur la théorie mathématique des relations
- Exple précédent traité avec le modèle relationnel :

- Relation enseignant

Code-e	Nom-e	Prenom-e	categ
E1	Tanin	M.	Prof
E2	Ans	J.C.	MCF
E3	Decaux	F.	agreg
E4	Carik	J.M.	certif
E5	Bage	R.	vacat

Relation matiere

Code-m	Lib-m	Coef-m
M1	Info	6
M2	gest	3
M3	math	6
M4	compta	3
M5	mathFi	1

- Relation E-M

Code-e	Code-m	Nb-heues
E1	M1	9
E1	M2	1
E2	M3	13
E2	M2	5
E2	M4	4
etc ..	etc..	etc..

Commentaires

- La relation enseignant : possède un nom,

  possède 4 attributs (degré 4), a des tuples (des lignes de valeurs), possède une clé primaire num-e (dont les valeurs distinguent les tuples entre eux), …
- il n ’y a pas 2 lignes identiques,
- il n ’y a pas 2 colonnes identiques,
- l ’ordre des colonnes (attributs) n ’est pas important, …

- **Exemple de manipulation des ces relations** (on utilisera le langage SQL, étudié en détail par la suite) => repose sur les opération ensemblistes (produit cartésien, projection, …).
- Q1. Libellés des matières enseignées par l ’enseignant de code « E3 »
- => 2 opérations ensemblistes
- a. dans E-M : tuples où code-e = « E3 » ==> relation R1
- b. jointure ou connexion entre R1 et matière sur un domaine (ici code-e)
-
- ==> on obtient le résultat

R1

Code-e	Code-m
E3	M2
E3	M4
E3	M5

Code-e	Code-m	Lib-m
E3	M2	Gest
E3	M4	compta
E3	M5	mathFi

Exemple (suite)

- Q2. Enseignants de la matière de code « M1 »
- => 2 opérations ensemblistes
- a. dans E-M : tuples où code-m = « M1 » ==> relation R1
- b. jointure ou connexion entre R1 et enseignant sur un domaine (ici code-m)
- ==> on obtient le résultat

R1

Code-e	Code-m
E1	M1
E4	M1
E5	M1

résultat

Code-e	Code-m	Nom-e	Prenom-e
E1	M1	Talin	M.
E4	M1	Carik	J.M.
E5	M1	Bage	R.

Quelques commentaires :

- simples d ’utilisation : on ne connaît pas comment sont stockées les données sur les supports.

- questions ‘ symétriques ’ => réponses symétriques.

- opérations basées sur l ’algèbre relationnelle,

Quelques éléments sur les SGBD Objets

- Fusion de 2 domaines : les BD et les lang. Objets.
- Intègrent aussi les techniques d ’interface H/M, des syst. distribués, …
- un SGBDOO repose sur la notion d ’objet . Il offre les fonctionnalités d ’un SGBD, et celle d ’un lang. Objet complet.) => combine les aspects statiques (description des structures) et dynamiques (pgms).
- Règles pour un SGBDOO : (en plus des règles pour les SGBD en général). Il doit :
- - permettre l ’encapsulation (cacher la complexité) des objets et le passage de messages,
- - permettre de définir et manipuler des objets composites,
- - offrir les notions de classes, types et méthodes,
- - offrir la notion d ’héritage,
- - permettre la surcharge des noms de méthodes et liens dynamiques (pour réécrire le code d ’une méthode héritée). En invoquant la méthode, un code ou un autre est exécuté => lien dynamique (exemple : billet -> train ou avion)
- - offrir un langage de programmation complet
- - être extensible

Les SGBD Objets (2)

- Un objet : (identificateur, valeur d ’un certain type, opérations ou méthodes applicables)
- Exemple classe : personne
  - attributs nom:string, date_naiss: integer, adresse: string
  - opérations : lirenom, majnom, majdate, majadresse, age
- traduction en langage d ’un SGBDOO : O2
- *class personne*
- *type tuple (nom : string,*
- *date_naiss : integer,*
- *adresse : string)*
- *method*
- *lirenom : string,*
- *majnom(n : string) : personne,*
- *majdate(d : integer) : personne,*
- *majadresse(nouvelle : string) : personne,*
- *age : integer*
- *end*

Les SGBD Objets (3)

- *Une valeur (instanciation) de la classe personne :*
- *tuple (nom : « Martin »,*
- *date_naiss : 071149*
- *adresse : « 10, rue vigne Paris »)*
- *ENCAPSULATION ==> impossible d 'accéder directement aux valeurs d 'attributs (=> privés). On y accède à travers les méthodes.*
- *Création d 'instances : p = new personne*
  - *p -> nom: accéder à la valeur de l 'attribut (on ne tient pas compte ici de l 'encapsulation)*
  - *p -> lirenom : appliquer une méthode à un objet*
- *p -> majnom(« Marc ») : créer une nouvelle (instance de) personne.*
- *Valeurs simples : integer, boolean, …*
- *Valeurs composites : utilisation des constructeurs TUPLE, SET, LIST*
- *Exemple d 'interrogation d 'une BDO sous le SGBDOO O2:*

SELECT ***h -> nom*** *FROM* ***v*** *IN* ***Les_villes***, ***h*** *IN* ***v_hotels***

WHERE ***v -> nom=« Londres »*** *AND* ***h -> etoiles = 2***

=> c 'est une requête pour trouver les noms des hôtels « 2 étoiles » à Londres.

Les SGBD Objets : Commentaires

- SGBD objets ne sont pas encore généralisés dans le commerce,
- pas de standard actuellement (ni conception, ni langages de manipulation, ..)
+ cumule les avantages des SGBD (notamment relationnels) et ceux des langages objets (héritage, encapsulation, …)

Les Systèmes de Gestion de Bases de Données (SGBD)

Le Modèle Entité/Association

Modèle Entité/Association

Représentation explicite de 3 concepts principaux: entité, association, d'attribut.

1. **Entité** = classe générique d'individus ou d'objets ayant les mêmes caractéristiques.
   Ex: les entités "clients", "produits" ou "fournisseurs", dans une base de données de magasin.

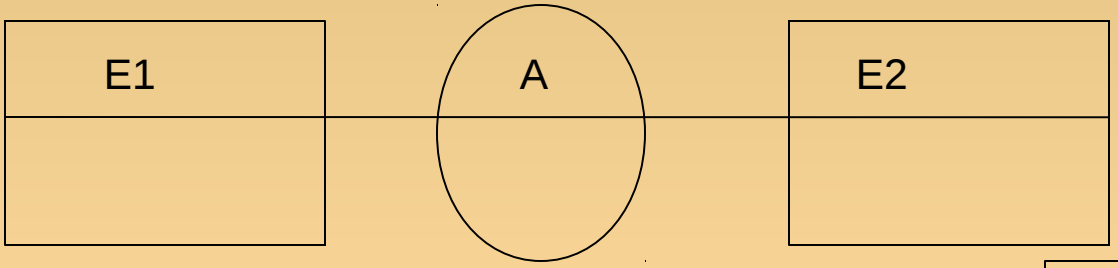
2. **Association** = classe générique de liens reconnus ou possibles entre individus ou objets.
   Ex: l'association "achète" lie les clients et les produits d'un magasin.

3. **Attribu**t = propriété distinctive d'une entité ou d'une association.
   Ex: le nom d'un client est un attribut de l'entité "clients".

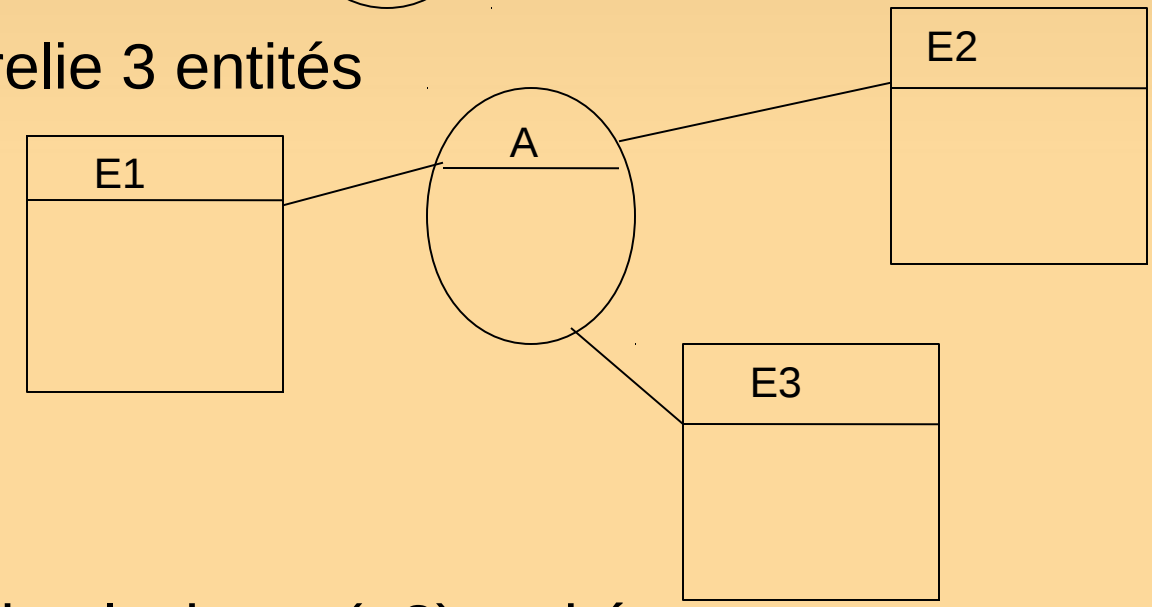
Rmq : Termes officiels : entité-type ou type-entité au lieu de entité, etc …
Par abus de langage, on parle de : entité, association, attribut

Association binaire/n-aire

- binaire : relie 2 entités

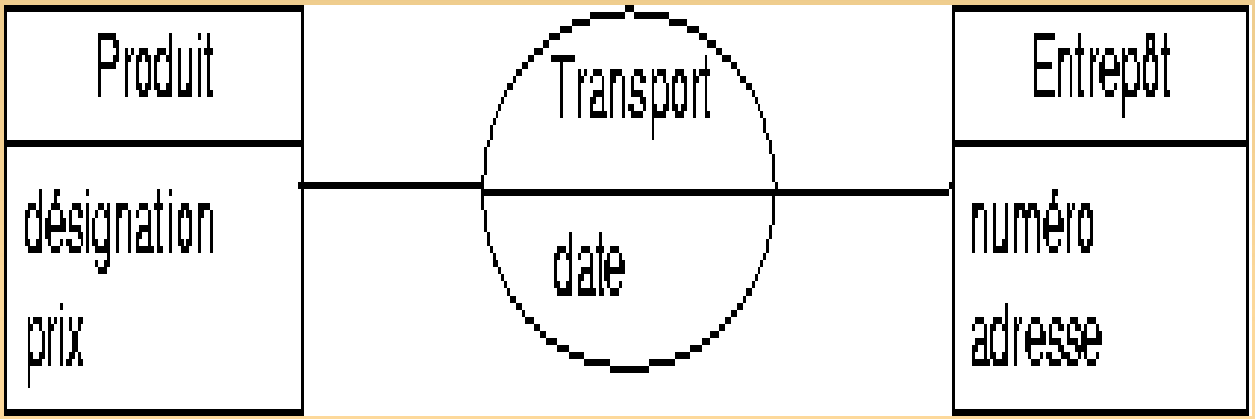


- ternaire : relie 3 entités



- n-aire : relie plusieurs (>2) entités

Exemple


Produit
désignation
prix
Transport
date
Entrepôt
numéro
adresse

Autres concepts : identifiant ou clé

- **Identifiant** : Un identifiant d ’une entité est constitué par un ou plusieurs de ses attributs dont les valeurs doivent identifier de manière unique cette entité.

- Au niveau d ’une entité, chaque attribut possède un domaine qui définit l'ensemble des valeurs possibles qui peuvent être choisies pour lui (entier, chaîne de caractères, booléen…).

- Exemples d'identifiant : le numéro d'immatriculation d'une voiture, le code-barre d'un produit…

- Remarque : une entité peut avoir plusieurs identifiants possibles.

suite

- On parle parfois de clé plutôt que d'identifiant.
- Chaque entité possède au moins un identifiant, éventuellement formé de plusieurs attributs.
- L'identifiant, qu'il soit explicite ou non, d'une association doit être la concaténation des identifiants des entités liées
- On admet cependant un identifiant plus naturel, à condition qu'il ne soit qu'un moyen d'exprimer plus simplement cette concaténation.

Autre concept : cardinalité d ’un couple (E,A)

- Définition : La **cardinalité** d'une association est le nombre de fois minimal et maximal qu'une entité peut intervenir dans une association de ce type.
- La cardinalité minimale doit être inférieure ou égale à la cardinalité maximale.
  - Exemple de cardinalité : un client peut commander entre 1 et n produits.

  ==> représentation : (1,n)
  - L'expression de la cardinalité est obligatoire pour chaque patte d'un type-association.

Autre concept : cardinalité d ’un couple (E,A), suite

- La cardinalité minimale peut-être :

- * 0 Cela signifie qu'une entité peut exister tout en n ’étant impliquée dans aucune association.
- * 1 Cela signifie qu'une entité ne peut exister que si elle est impliquée dans au moins une association.
- * n Cela signifie qu'une entité ne peut exister que si elle est impliquée dans plusieurs associations.

- Attention ! Le cas *n* comme cardinalité minimale est rare et pose problème. Il est prudent de l'éviter.

Autre concept : cardinalité d ’un couple (E,A), suite

- La cardinalité maximale peut-être :

- * 1 Cela signifie qu'une entité peut être impliquée dans au maximum une association.
- * n Cela signifie qu'une entité peut être impliquée dans plusieurs associations.

- Attention ! En toute logique, la cardinalité max 0 ne doit pas exister : il démontre un problème de conception puisque l ’entité est inutile à l ’association. Il faut alors reconsidérer la cardinalité ou retirer la liaison entre l ’entité et l ’association.

Remarques

- * Une entité ou une association est composée d ’ occurrences ou instances.
- * Un attribut ne peut en aucun cas être partagé par plusieurs entités ou associations.
- * Il est parfois difficile de faire un choix entre une entité et une association : un mariage est-il une association entre deux personnes ou une entité pour laquelle on veut conserver un numéro, une date, un lieu, etc. et que l'on souhaite manipuler en tant que telle ? Le contexte doit aider à répondre à ce genre de question.
-

Remarques (2)

- * Lorsqu'on ne parvient pas à trouver d'identifiant pour une entité, il faut se demander s'il ne s'agit pas en fait d'une association. Si ce n'est pas le cas, un identifiant arbitraire numérique entier peut faire l'affaire.
- * Lorsque toutes les pattes d'une association portent la cardinalité 1,1, il faut se demander si cette association et les entités liées ne décrivent pas en fait une seule entité.
- * Pour faciliter la lecture du schéma, il est assez courant de ne pas y faire figurer les attributs ou de ne conserver que ceux qui font partie des identifiants. Les attributs cachés doivent alors absolument être spécifiés dans un document à part.

Autre concept : **type** d ’association

Il existe différents types d'associations:

* **Association "1 - 1"**: A une occurrence de l'entité E1 peut correspondre au plus une occurrence de l'entité E2, et vice versa.
Ex: l'association "loue" liant deux entités "clients" et "voitures" d'une agence de locations de voitures, si un client ne peut louer qu'une voiture à la fois.

* **Association "1 - n"**: A une occurrence de l'entité E1 peuvent correspondre plusieurs occurrences de l'entité E2,
mais à une occurrence de l'entité E2 peut correspondre au plus une occurrence de l'entité E1.
Ex: l'association "possède" liant deux entités "clients" et "comptes" d'une banque

* **Association "m - n"**: A une occurrence de l'entité E1 peuvent correspondre plusieurs occurrences de l'entité E2, et vice versa.
Ex: l'association "achète" liant deux entités "clients" et "produits" d'un supermarché.

Problèmes de modélisation

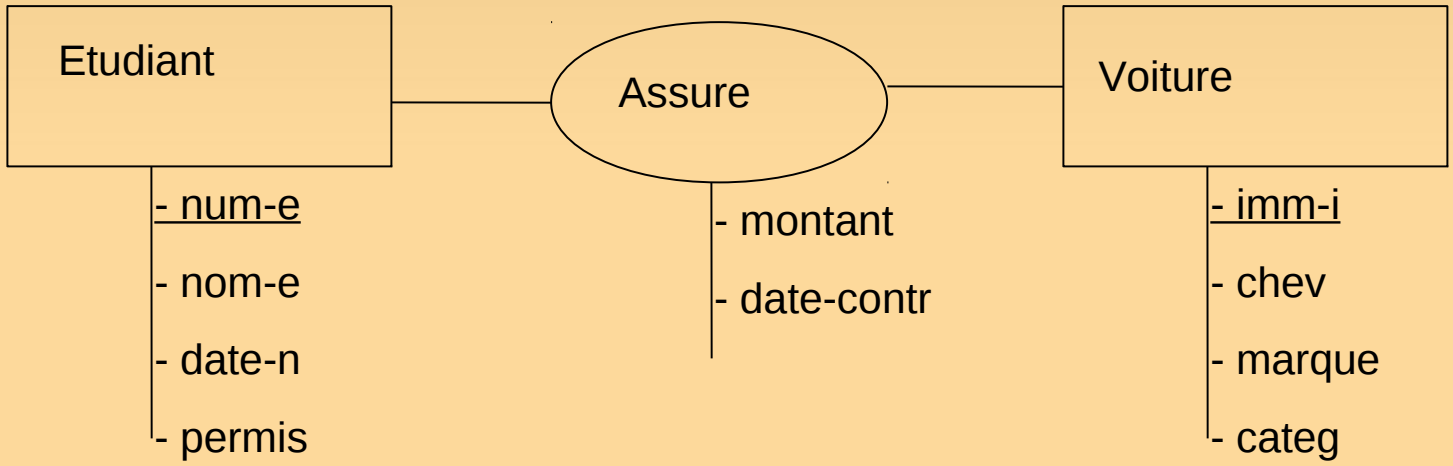
- Une fois qu ’un objet est reconnu comme entité, il ne peut pas être à la fois une association, et vice-versa.
- Une association lie forcément 2 objets (ou plus). Une entité est forcément liée à (au moins) une autre entité par (au moins) une association.
- Une fois qu ’un attribut est placé sous une entité, il ne peut pas appartenir à la fois à une association, ou à une autre entité, et vice-versa.

Problèmes de modélisation (2)

- Le locuteur ne se modélise pas lui-même : l ’organisme du point de vue duquel on réalise le modèle ne fait pas partie du modèle.
- On ne représente une entité (association) que si l ’on sait reconnaître une classe d ’individu dans cette entité (association).
- La détermination des attributs pertinents est fonction du point de vue. Ex : la tailles d ’un étudiant n ’est pas utile pour la scolarité, mais est pertinente pour ses papiers d ’identité.
- Un attribut appartient à une association si la connaissance des valeurs des clés des entités reliées par cette association détermine de manière unique la valeur de cet attribut.)

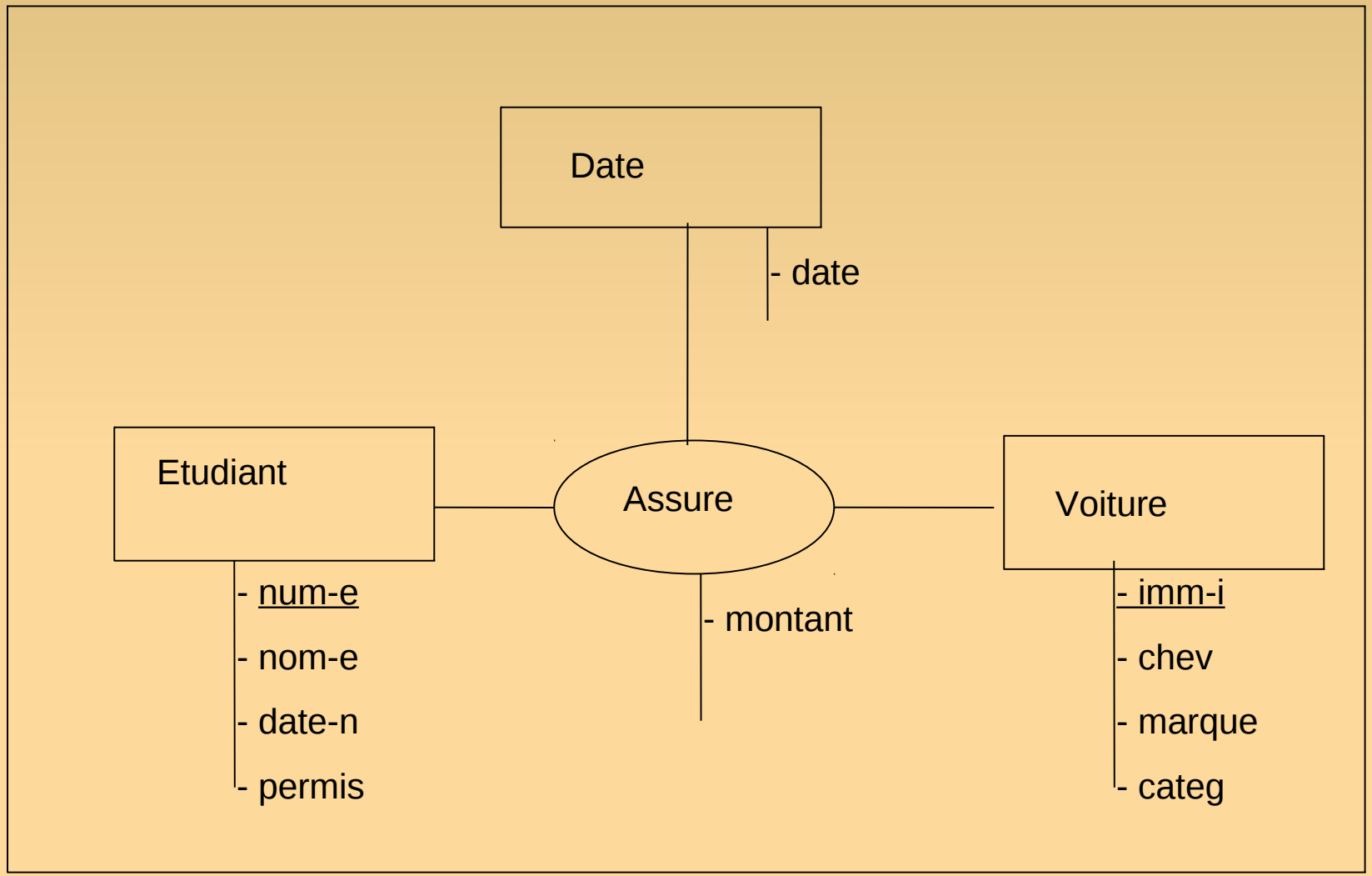
Prise en compte du temps

- 1. Point de vue synchronique (instantané) : le temps n ’intervient pas comme élément discriminatoire.
- Exemple : On ne peut pas faire l'historique des assurance de la même voiture, du même propriétaire.



- 2. Point de vue diachronique (historique).

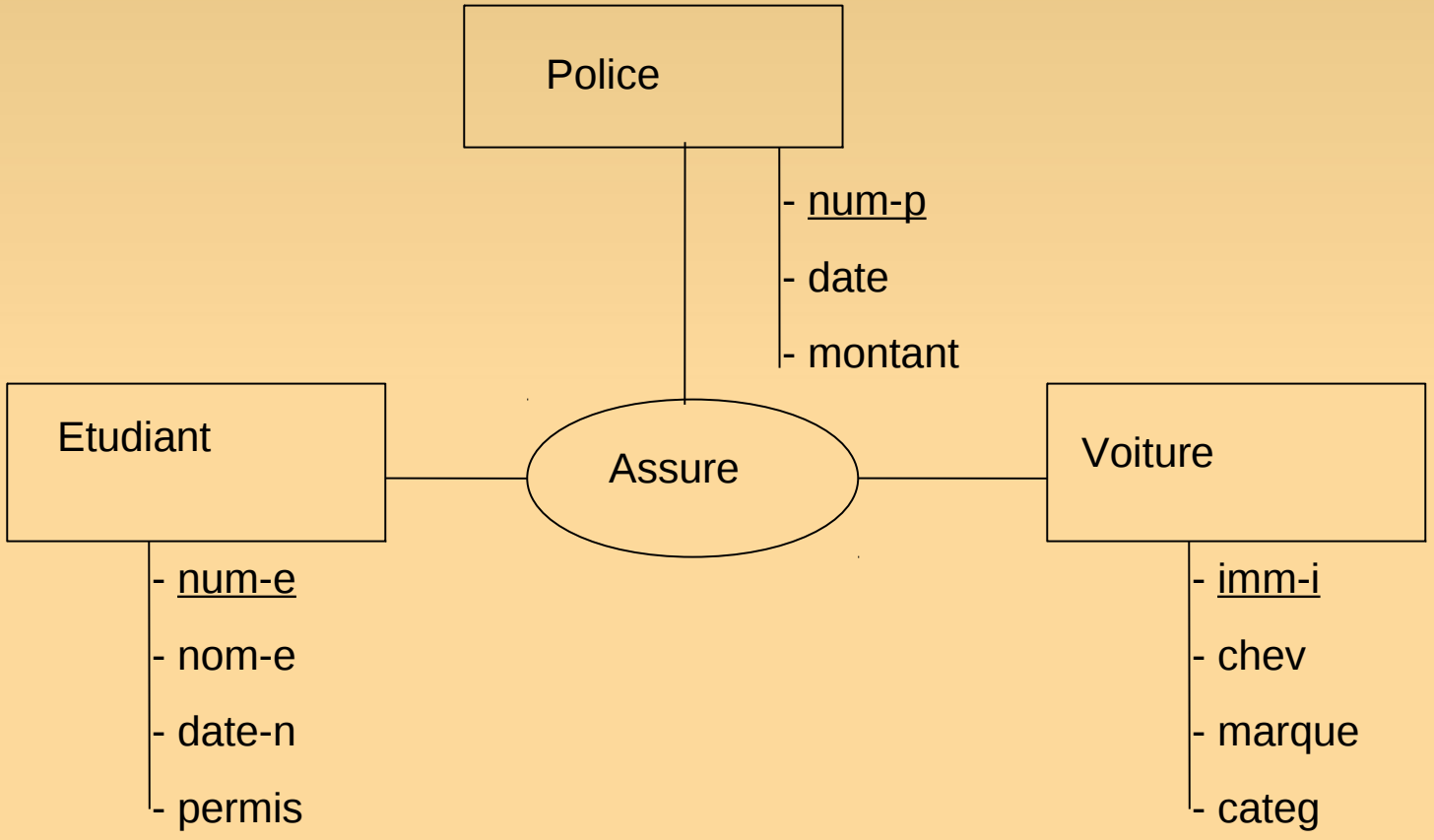
2.1. Création d ’une entité date : une assoc. n-aire devient (n+1)-aire



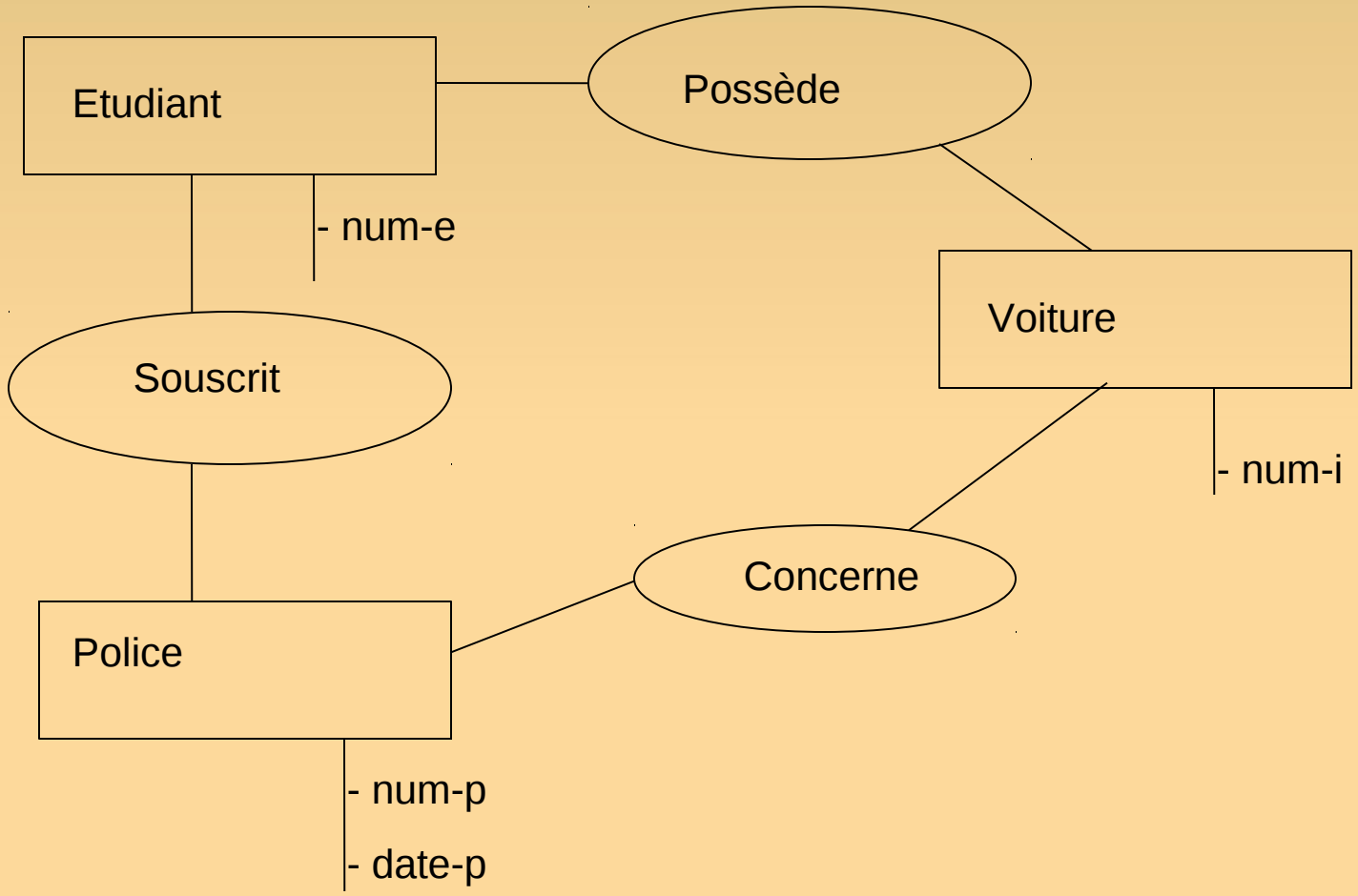
- On peut faire l'historique :
- proprio p1, voit. V1, date d1 : assurace 1000 euros
- Proprio p1, voit. V1, date d2 : assurance 900 euros; etc.

2.2 Point de vue diachronique (historique). Introduction d ’événements datés

Assurance => police d ’assurance (document avec date, …)



On en fait des associations binaires …



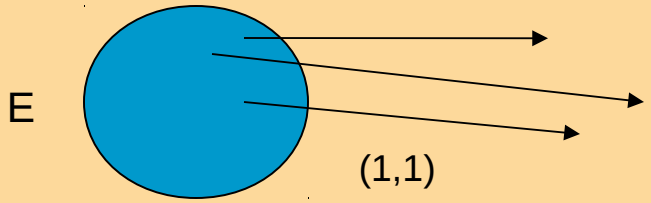
Passage E/A --> Relationnel

Passage modèle E/A --> modèle relationnel

Modèle E/A : notion d ’entités, attributs, associations, contraintes de cardinalités
Modèle relationnel : notion de relations, attributs, dépendances fonctionnelles
Les df dans le modèle E/A :

SUR L ’ ENTITE : une clé d ’une entité (identifiant) est un ensemble d ’attributs qui détermine fonctionnellement tous les autres attributs de l ’entité.

SUR LES CARDINALITES : si la cardinalité du couple (E, A) est (0,1) ou (1,1) alors une occurrence de l ’entité E détermine les occurrences de toutes les autres entités intervenant dans l ’association A.



Les clés des autres entités dépendent fonctionnellement de la clé de E

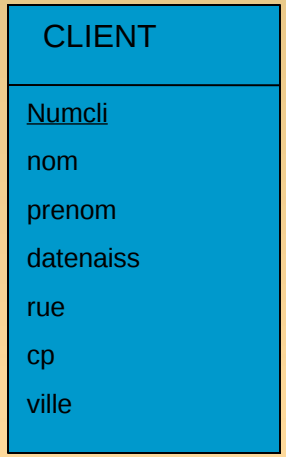
Passage modèle E/A --> modèle relationnel

Si T est un attribut dans d ’une association dans laquelle les cardinalités sont toutes (0,n) ou (1,n), alors T est déterminé par les occurrences de toutes les entités intervenant dans la définition de l ’association.=> T dépend fonctionnellement des clés des entités intervenant dans l ’association.

D ’où les règles de passage suivantes :

Passage modèle E/A --> modèle relationnel

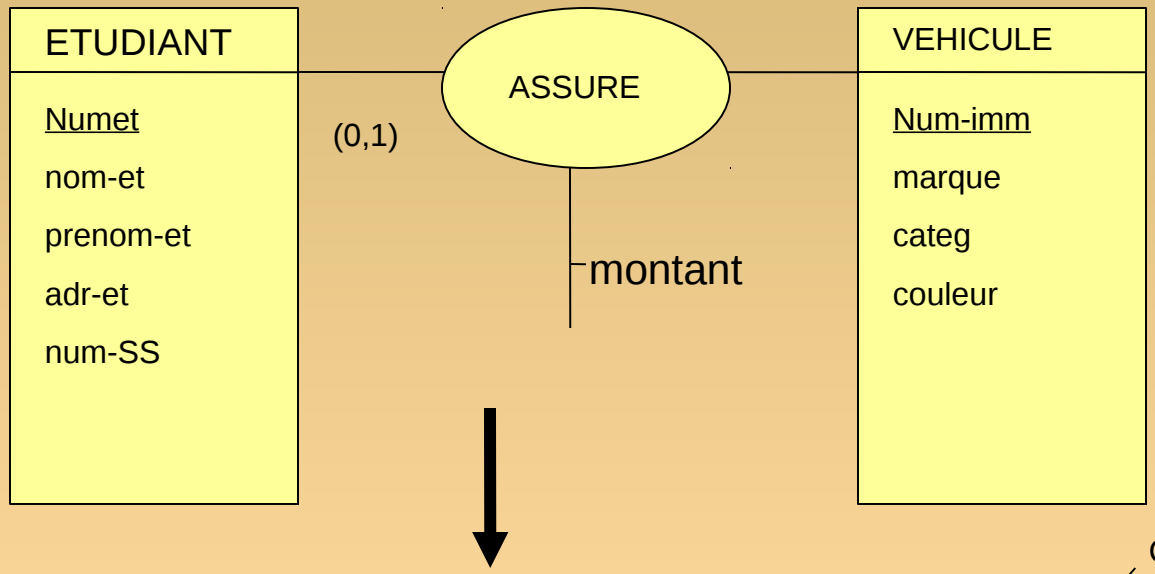
R1. Chaque entité devient une relation. Les attributs de l'entité deviennent attributs de la relation. L 'identifiant de l 'entité devient clé de la relation.



CLIENT(numcli, nom, prénom, datenaiss, rue, cp, ville)

R2. Si dans une association, il existe une entité pour laquelle la cardinalité est (0,1) ou (1,1), alors ajouter à la relation représentant cette entité les identifiants des autres entités intervenant dans cette association comme clé étrangère, ainsi que les attributs éventuels de l' association.

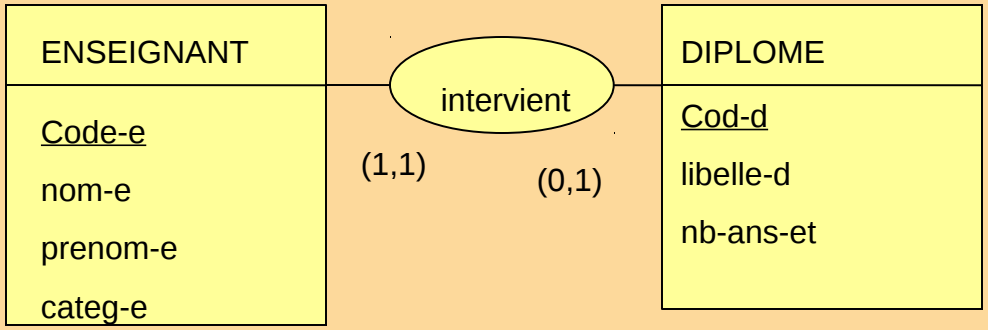
Exemple :



Clé étrangère

ETUDIANT(num_et, nom-et, prenom-et, adr-et, num-SS, **num-imm**)

Remarque : si on a des cardinalités (0,1) et (1,1), privilégier la cardinalité (1,1).

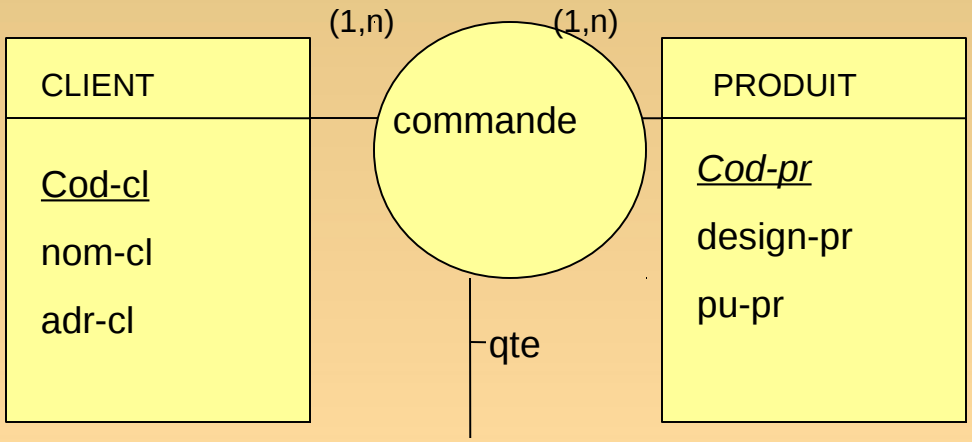


ENSEIGNANT(code-e, nom-e, prenom-e, categ-e, **cod-d**)

DIPLÔME(code-e, libelle-e, nb-ans-et)

R3. Toute association où il n ’existe pas de cardinalité (0,1) ou (1,1), devient une relation avec comme clé la concaténation des clés des entités qu’elle relie (relations participantes) et comme attributs les attributs éventuels de l’association.

Exemple :



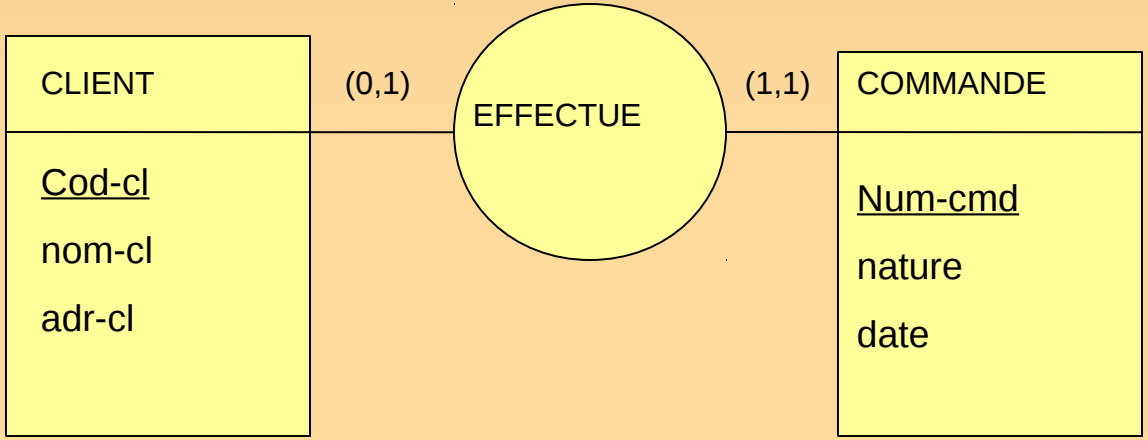
➡ CLIENT(Cod-cl, nom-cl, adr-cl)

PRODUIT(*Cod-pr*, design-pr, pu-pr)

COMMANDE(cod-cl, cod-pr, qte)

Remarques

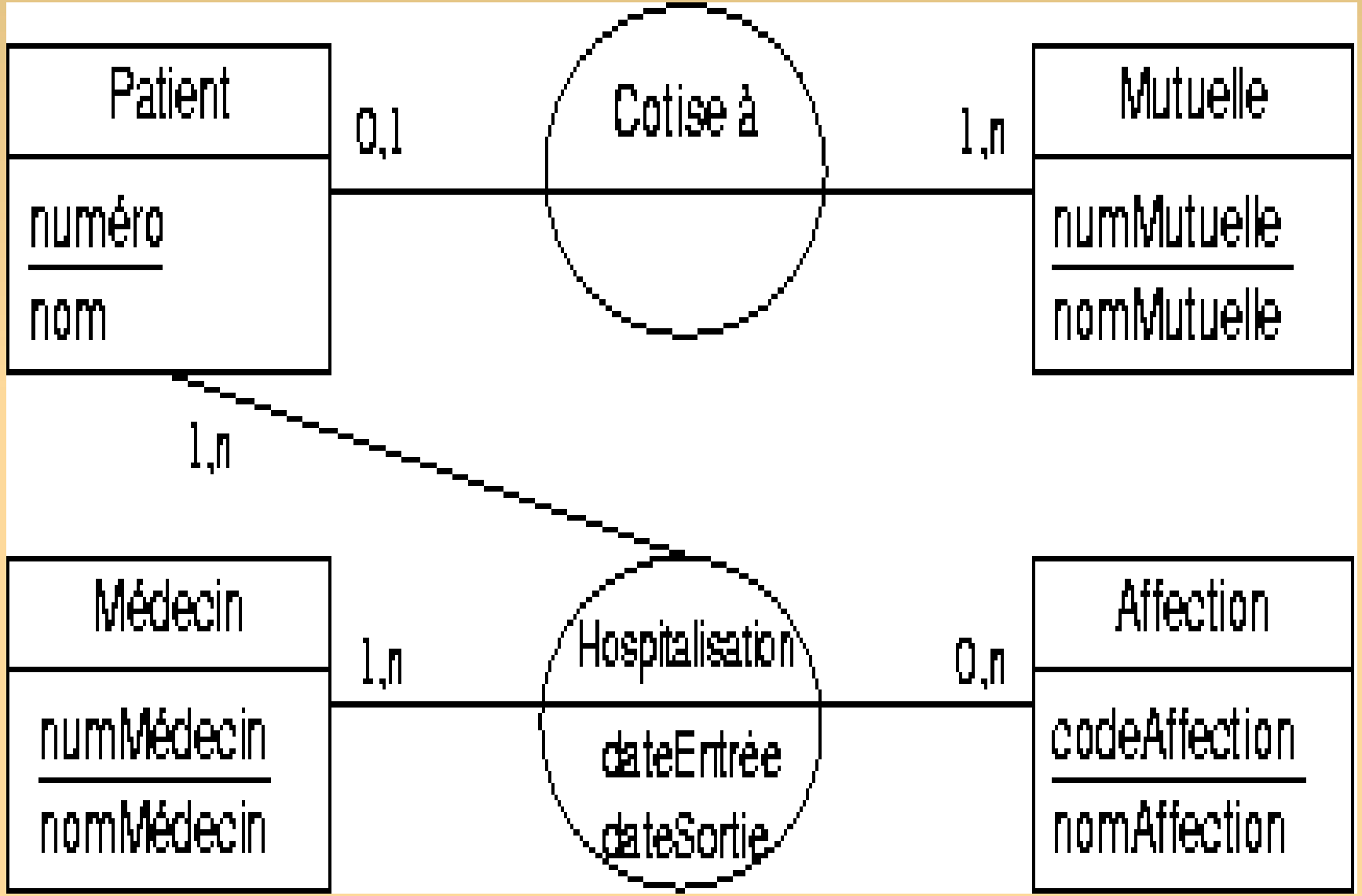
- Si, pour une association, il existe des entités pour lesquelles la cardinalité du couple E/A est (0,1) et d ’autres pour laquelle elle est de (1,1), alors privilégier la cardinalité (1,1).
- Exemple :



➡ CLIENT (cod-cl, nom-cl, adr-cl)

COMMANDE (Num-cmd, nature, date, **code-cl**)

Modèle E/A simplifié



Relations déduites de ce schéma :

- * Patient(numéro, nom, numMutuelle)
- * Mutuelle(numMutuelle, nomMutuelle)
- * Médecin(numMédecin, nomMédecin)
- * Affection(codeAffection, nomAffection)
- * Hospitalisation(numéro, codeAffection, numMédecin, dateEntrée, dateSortie)

- Notation : nomRelation (clé, Attribut1, Attribut2, …)

Langages relationnels - L’algèbre relationnelle

Les langages relationnels :

- sont utilisés pour effectuer des requêtes sur une BD relationnelle

- utilisent 2 approches qui expriment les mêmes opérations :
  - algèbre relationnelle
  - calcul relationnel

Les opérateurs de l ’algèbre relationnelle sont des opérateurs ensemblistes.

Un opérateur prend en entrée une ou deux relations (ensembles de tuples de la base) et retourne un résultat qui est également une relation.

Il existe 5 opérateurs de base :

- les opérateurs unaires : selection et projection
- les opérateurs binaires : union, différence et produit cartésien

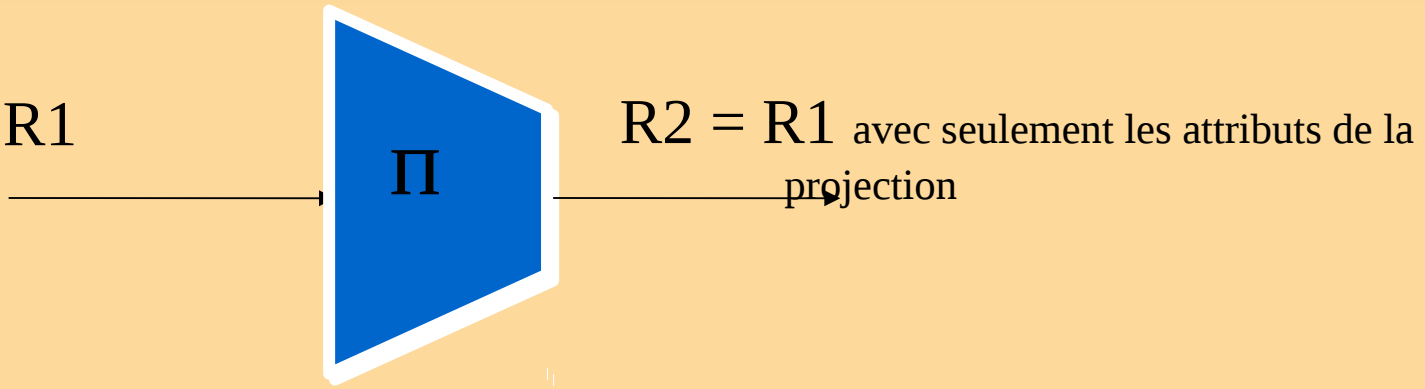
D ’autres opérateurs existent qui peuvent s ’exprimer à l ’aide des opérateurs de base. Ce sont la jointure, la division et l ’intersection

La projection : Π

Projection des attributs d ’une relation R sur un sous-ensemble des attributs de R :

$\Pi_{A1, A2, \ldots, Ak}(R) = R(A1, A2, \ldots, Ak)$

où A_1, A_2, …,A_k sont un sous-ensemble du schéma de la relation R (égal ou inclus).



La projection sur A1, A2, …, Ak élimine tous les autres attributs de la relation et supprime les tuples dupliqués.

Exemple de projection

R

X	Y	Z
a	b	c
d	a	b
c	b	d
a	b	e
e	e	a

$=>\Pi_{X,Y}(R) =$

R'

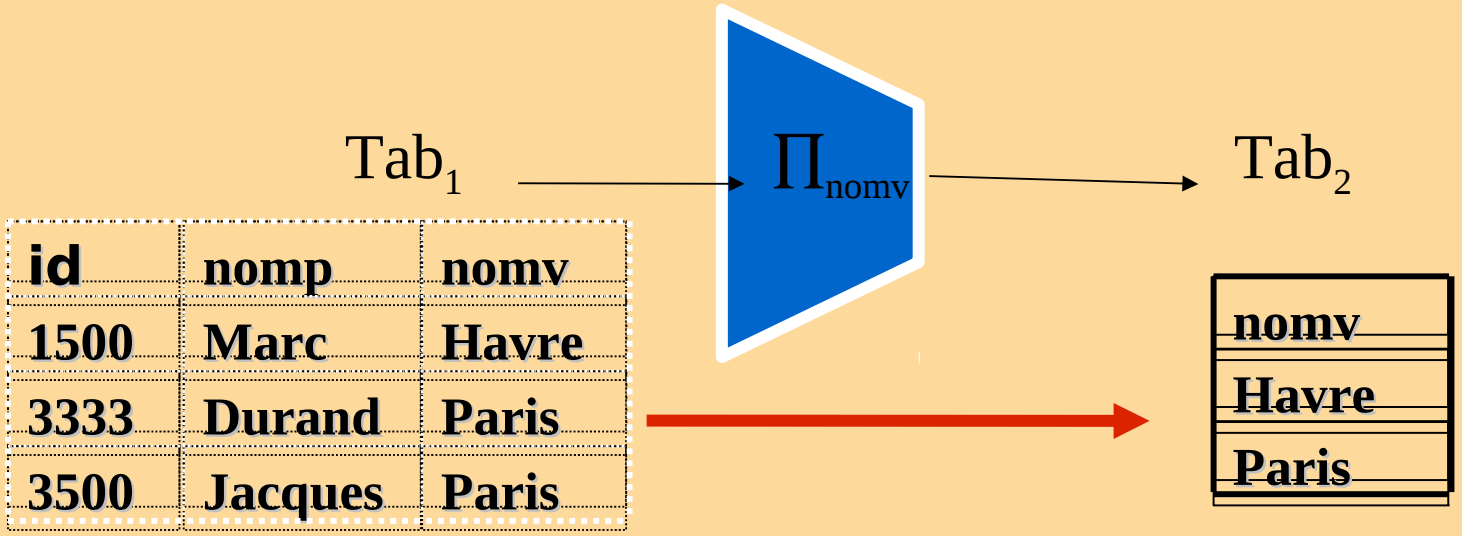
X	Y
a	b
d	a
c	b
e	e

R = (X,Y,Z) et R' = $\Pi_{X,Y}(R) =$ projection de R sur les attributs X et Y

Exemple de projection (2)

- Requête : Soit la relation Ville (id, nomp, nomv)
  - Quels sont les villes de résidence des personnes de la base (projection sur l'attribut nomv)

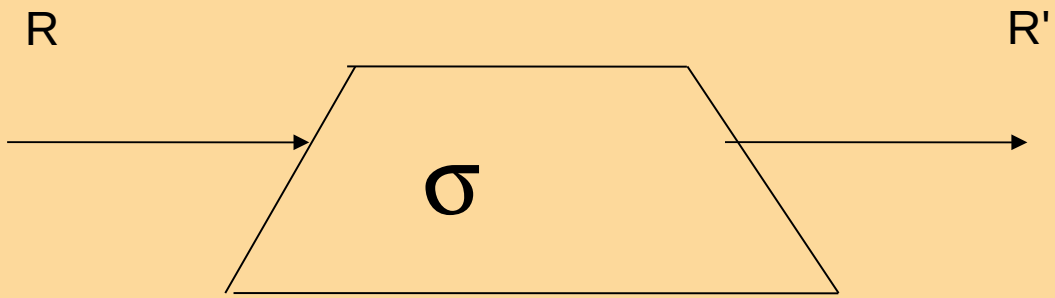
$$\mathbf{Tab2 = \mu_{nomv}(Tab_1)}$$



Selection (restriction)

La selection se fait en fonction d'une condition C portant sur des attributs de R. Le résultat est une relation dont les attributs satisfont la condition.

On note : $\sigma_C(R)$



Exemple de selection

R

X	Y	Z
a	b	1
d	a	2
c	b	3
a	b	4
e	e	5

$\Rightarrow \sigma_{Y='b'}(R) = R'$

R'

X	Y	Z
a	b	1
c	b	3
a	b	4

R

X	Y	Z
a	b	1
d	a	2
c	b	3
a	b	4
e	e	5

$\Rightarrow \sigma_{C}(R) = R'$

où C : (X='a' $\vee$ Y='a') $\wedge$ Z$\leq$ 3

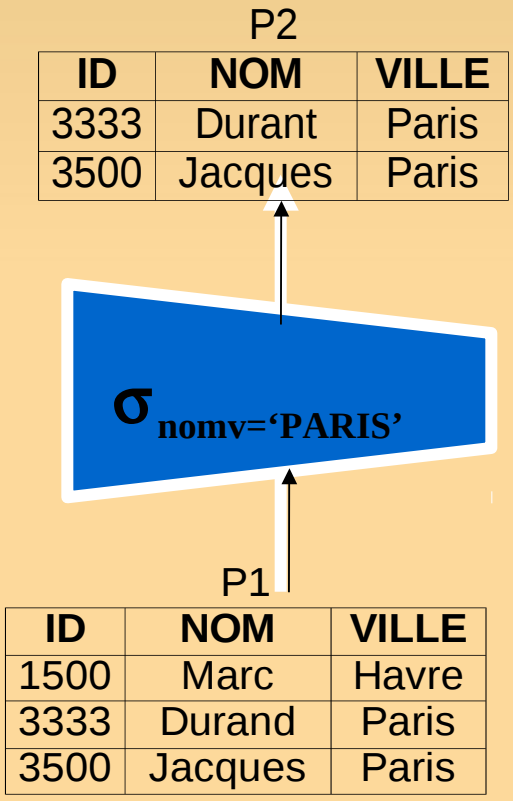
R'

X	Y	Z
a	b	1
d	a	2

Exemple de sélection (2)

- Requête :
  - Sélectionner tous les individus habitant à Paris.

$$\mathbf{Tab1_2} = \sigma_{\text{vomv='Paris'}}(\text{Tab1})$$



Les conditions de sélection

Il s ’agit d ’une formule logique qui relie, par des connecteurs logiques (AND, OR, NOT), des expressions de la forme :

A_i **op** A_j ou (A_i **op** a) ou A_i

où A_i, A_j sont des attributs de la relation R

'a' est un élement (une valeur) du domaine de A_i

op est un opérateur de comparaison : =, <, >, <=, >=, <>

Expressions de l ’algèbre relationnelle

- L ’algèbre relationnelle est fermée par rapport aux opérations de l ’A.R. : le résultat d ’une opération relationnelle est aussi une relation.

- Les opérations relationnelles peuvent être combinées et former des expressions plus complexes.
- Ex : Relation R = Commande (nom, prenom, nomc, qte)

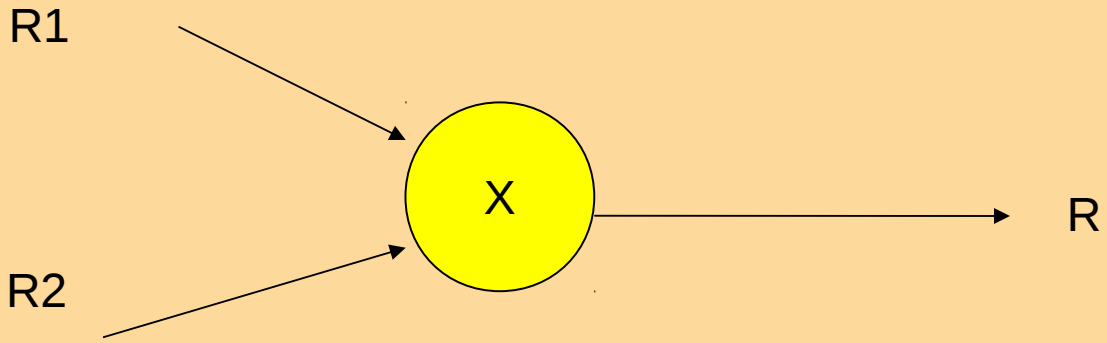
R’ = $\pi_{pnom}(\sigma_{nom}(‘paul’(commande))$

R’’ = $\sigma_{nom}(‘paul’(commande))$: contient les commandes de ‘paul’

Produit cartésien

- Produit cartésien de la relation R1 par la relation R2 : R1 x R2
- Argument : 2 relations quelconques

  R1 (A1, A2, …, An) et R2 (B1, B2, …, Bk)
- Schéma de la relation résultat T : R1 x R2 : (A1, …, An, B1, …, Bk)
- Les occurrences de R : ensemble des tuples ayant n+k attributs :
  - dont les n valeurs des premiers attributs sont les tuples de R1
  - et les k dernieres sont les tuples de R2



Exemple

R

A	B
1	1
1	2
3	4

S

C	D	E
a	b	a
a	b	c
b	a	a

R x S

A	B	C	D	E
1	1	a	b	a
1	2	a	b	a
3	4	a	b	a
1	1	a	b	c
1	2	a	b	c
3	4	a	b	c
1	1	b	a	a
1	2	b	a	a
3	4	b	a	a

Jointure naturelle

- Soient 2 relations R et S ayant des attributs en commun

  $R(A_1, \ldots, A_m, \mathbf{X_1, \ldots, X_K})$

  $S(B_1, \ldots, B_n, \mathbf{X_1, \ldots, X_k})$

- Schéma de la relation R ⋈ S, jointure naturelle de R et S :

  $T(A_1, \ldots, A_m, B_1, \ldots, B_n, \mathbf{X_1, \ldots, X_K})$

Un tuple de R ⋈ S comporte donc (m+n+k) attributs.

R

A	B	C
a	b	c
d	b	c
b	b	f
c	a	d

S

B	C	D
b	c	d
b	c	e
a	d	b

R ⋈ S

A	B	C	D
a	b	c	d
a	b	c	e
d	b	c	d
d	b	c	e
c	a	d	b

Jointure naturelle (suite)

- La jointure naturelle correspond à un produit cartésien, suivi d ’une sélection.
- Soient 2 relations R et S ayant des attributs en commun

  $R(A_1, \ldots, A_m, X_1, \ldots, X_K)$

  $S(B_1, \ldots, B_n, X_1, \ldots, X_k)$. Soit $V = \{X_1, \ldots, X_k\}$

$R \bowtie S = \Pi_U (\sigma_{\forall A \in V : R.A=S.A} (R \times S))$

où U = l ’ensemble des attributs de R et S et

R.A est l ’attribut A de R

Jointure naturelle - algorithme

Début

Pour tout tuple a de R et tout tuple b de S :

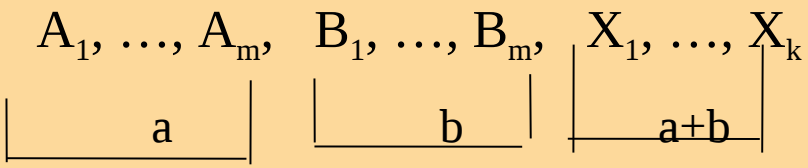
1. Concaténer a et b. On obtient un tuple avec comme attributs a | b, c-à-d :

$$A_1, \ldots, A_m, X_1, \ldots, X_k, B_1, \ldots, B_n, X_1, \ldots, X_k$$

2. Ne garder ce tuple que si chaque attribut X_i de a est égal à l'attribut X_i de b :

$$\forall_{i=1::..k}\ a.X_i = b.X_i$$

3. Eliminer les valeurs (colonnes) dupliquées. On obtient, pour la jointure naturelle, un tuple avec comme attributs :



θ-jointure

- Notée : . $R \bowtie_{Ai\ \theta\ Bj} S$, où $\theta \in \{=, , < >, \leq, \geq, \neq\}$
- C ’est une jointure entre 2 relations R et S avec :
- $R = (A_1, \ldots, A_m)$, $S = (B_1, \ldots, B_n)$
- Schéma de $T = R \bowtie_{Ai\ \theta\ Bj} S = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
- La valeur de T est : $\sigma_{Ai\ \theta\ Bj}$ (RxS) : sélection des tuples de RxS tels que $A_i\ \theta\ B_j$
- Equijointure : on parle de équijointure quand l’opérateur θ est l ’égalité.

Exemple de θ-jointure

$R \bowtie S$

R

A	B
1	a
1	b
3	a

S

C	D	E
1	b	a
2	b	c
4	a	a

RxS

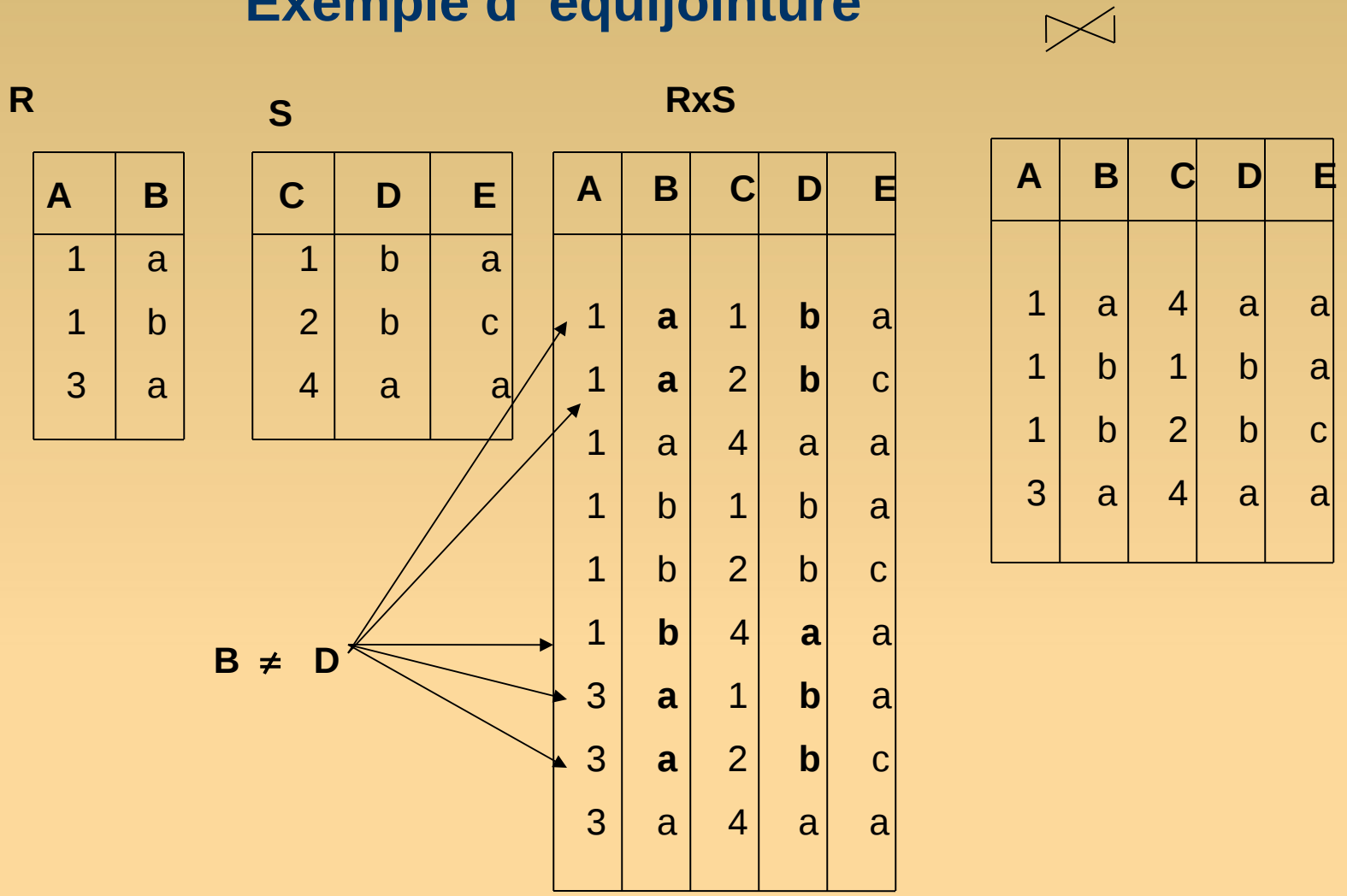
A	B	C	D	E
1	a	1	b	a
1	a	2	b	c
1	a	4	a	a
1	b	1	b	a
1	b	2	b	c
1	b	4	a	a
3	a	**1**	b	a
3	a	**2**	b	c
3	a	4	a	a

A > C →

A > C →

A	B	C	D	E
1	a	1	b	a
1	a	2	b	c
1	a	4	a	a
1	b	1	b	a
1	b	2	b	c
1	b	4	a	a
3	a	4	a	a

Exemple d ’équijointure



Exercices : Equoijinture, jointure naturelle

- Soient les relations :

Immeuble (Adr-Imm, NB-etages, Date-Const, Proprio)

App-Imm (Adr-Imm, Num-App, Etage)

1. Nom du propriétaire de l ’immeuble où est situé l ’appartement occupé par *Dupond* :
   jointure naturelle

$\Pi_{Proprio}$ (Immeuble $\bowtie$ $\sigma_{Occup = «Dupond»}$(App-Imm))

Jointure naturelle

2. Appartements occupés par des propriétaires d’immeubles :

$\Pi_{Adr\text{-}Imm,\ Num\text{-}App,\ Etage}$ (App-Imm $\bowtie_{Occup=Proprio}$ Immeuble))

équijointure

Autre exemple de requête

- Soit le schéma :

  Commandes (pnom, cnom, num-cmde, qte)

  Clients (cnom, cadr, balance)

Nom et adresse des clients qui ont commandé des briques :

$\Pi_{cnom,cadr}$ (Clients $\bowtie$ $\sigma_{pnom = «briques»}$(Commandes))

Jointure naturelle

UNION

- Soient 2 relations $R(A_1, \ldots, A_m)$ et $S(A_1, \ldots, A_m)$
- Le schéma de $T = R \cup S$ est : $T(A_1, \ldots, A_m)$
- Les tuples de T : union ensembliste sur $D_1 x \ldots x D_m$ avec

  D_i domaine de A_i

Les doublons sont éliminés.

$T = R \cup S = \{t / t \in R \vee t \in S\}$

R

A	B
a	b
a	c
d	e

S

A	B
a	b
a	e
d	e
f	g

R ∪ S

A	B
a	b
a	c
d	e
a	e
f	g

DIFFERENCE

- Soient 2 relations $R(A_1, \ldots, A_m)$ et $S(A_1, \ldots, A_m)$
- Le schéma de $T = R - S$ est : $T(A_1, \ldots, A_m)$
- Les tuples de T : différence ensembliste sur $D_1 x \ldots x D_m$ avec

  D_i domaine de A_i

$$T = R - S = \{t / t \in R \wedge t \notin S\}$$

R

A	B
a	b
a	c
d	e

S

A	B
a	b
a	e
d	e
f	g

R - S

A	B
a	c

S - R

A	B
a	c
f	g

INTERSECTION

- Soient 2 relations $R(A_1, \ldots, A_m)$ et $S(A_1, \ldots, A_m)$
- Le schéma de $T = R \cap S$ est : $T (A_1, \ldots, A_m)$
- Les tuples de T : intersection ensembliste sur $D_1 x \ldots x D_m$ avec

  D_i domaine de A_i

$$T = R \cap S = \{t / t \in R \wedge t \in S\}$$

R ∩ S = R - (R - S)

(intersection)

A	B
a	b
d	e

R

A	B
a	b
a	c
d	e

S

A	B
a	b
a	e
d	e
f	g

R - S

(différence)

A	B
a	c

Semijointure

- La semijointure R ⋉ S correspond à une projection sur les attributs de R de la jointure naturelle entre R et S : Π_U (R ⋈ S)
- Si R = (A_1, …, A_m, **X_1, …, X_k**) et S = (B_1, …, B_n, **X_1, …, X_k**) alors le schéma de T = R ⋉ S = (A_1, …, A_m, **X_1, …, X_k**)
- **Exemple :**

R

A	B	C
a	b	c
d	b	c
b	b	f
c	a	d

S

B	C	D
b	c	d
b	c	e
a	d	b

=> R ⋈ S

A	B	C	D
a	b	c	d
a	b	c	e
d	b	c	d
d	b	c	e
c	a	d	b

jointure naturelle

=> R ⋉ S

A	B	C
a	b	c
d	b	c
c	a	d

semijointure

Division

R

A	B	C	D
a	b	x	m
a	b	y	n
a	b	z	o
b	c	x	o
b	d	x	m
c	e	x	m
c	e	y	n
c	e	z	o
d	a	z	p
d	a	y	m

S

C	D
x	m
y	n
z	o

R ÷ S

A	B
a	b
c	e

Division : exemple

COMM

Num	nom	p-nom	qte
1	**Jean**	**briques**	**100**
2	**Jean**	**ciment**	**10**
3	**Jean**	**plâtre**	**5**
4	Paul	briques	300
5	Paul	platre	8
5	Vincent	platre	15

PROD

P-nom
briques
ciment
plâtre

Client(s) qui ont commandé tous les produits

COMM ÷ PROD

nom
Jean

Division : formellement

- Soient R1(A_1, …, A_m, X_1, …, X_k) et S (X_1, …, X_k) c-à-d les attributs de S sont inclus dans R.
- T = R a pour schéma : T (A_1, …, A_m)
- Tuples de T = R ÷ S

  $=\{(a_1, \ldots, a_m) / \forall (x_1, \ldots, x_k) \in S : (a_1, \ldots, a_m, x_1, \ldots, x_k) \in R$

- La division peut s ’exprimer en utilisant les opérateurs *produit cartésien, projection et différence* : R ÷ S = R1 - R2 où :

$R1 = \Pi_{A1, \ldots, Am}(R)$ et $R2 = \Pi_{A1, \ldots, Am}((R1 \times S) - R)$

Opération de renommage

- Notée : ρ , c ’est une opération unaire.
- Consiste à remplacer le nom d ’un attribut par un autre.
- Soit R (A_1, …, A_m), on a :

  $\rho_{Ai \rightarrow Bi}$ R : T (A_1, …, A_{i-1}, $\mathbf{B_i}$, A_{i+1}, ..., A_m)

Les Systèmes de Gestion de Bases de Données (SGBD)

df - normalisation

Le modèle relationnel

df - normalisation

Etude du modèle relationnel

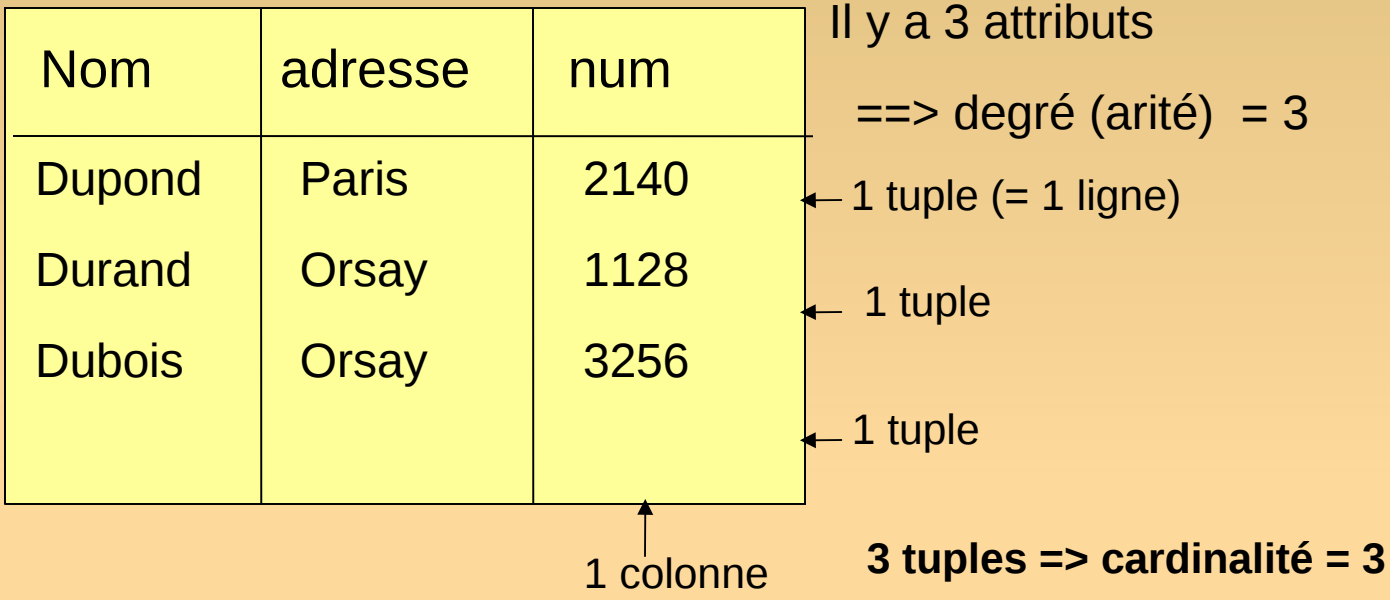
- Introduit par E.F. Codd en 1970.
- Le mieux formalisé (bases mathématiques)
  - Objectifs du modèle :
    - indépendance physique
    - traitement du problème de redondance des données
    - langages non procéduraux
    - devenir un standard de SGBD
  - Eléments du modèle :
    - domaine = ensemble de valeurs (prix dans]10, 30[)
    - attribut = variable avec valeurs dans un domaine (prix)
    - relation sur A_1, A_2, …,A_n

      de domaines respectifs D_1, D_2, …, D_n
      - = sous-ensemble du produit cartésien : $D_1 x D_2 x \ldots x D_n$.
      - = ensemble de tuples (x_1, x_2, ...x_n) où x_i est dans D_i, i=1,n
      - Une valeur spéciale (NULL) peut être prise par un attribut.

- Une bases de données relationnelles : est un ensemble de relations.
- Une relation R sur les domaines $D_1xD_2x…xD_n$ est représentée sous forme de table. Chaque ligne est un tuple (a1, a2, …, an) où chaque élément ai est dans Di.
- L ’ordre des lignes n ’est pas important.
- Les valeurs des colonnes de la tables sont les valeurs prises par les attributs.
- Le nombre d ’attributs d ’une relation est appelé *arité* de la relation.

Exemple

- Relation personne (nom, adresse, num)



Domaines:

nom : domaine de tous les noms de personnes (chaînes de car.)

adresse : domaine de toutes les adresses (chaînes de car.)

num : domaines de tous les numéros de personnes (des entiers naturels)

Quelques Contraintes d ’Intégrité (CI)

- *Contrainte d ’intégrité (CI)* : propriété que doit vérifier une relation = p(R)
- Clé primaire : attri. (ou ens d ’att) dont les valeurs permettent de distinguer les tuples les uns des autres dans une relation. (identifiant) (numetud pour ETUDIANT)
- Clé étrangère : attribut non clé qui est **clé primaire** d ’une autre relation.

  FOURNISSEUR (numf, nom, adr),

  PRODUIT(codep, lib, numf)

  On a : numf et codep = clés primaires, numf est clé étrangère de PRODUIT.

- Contrainte de domaine : condition sur les valeurs d ’attributs.

  Ex : prixunitaire >100 et ET prixunitaire <150

1. Attribut : identificateur (un nom) décrivant une information stockée dans une base. Par exemple, le numéroDeSécuritéSociale et le nom d'une personne sont des attributs.

2. Domaine : le domaine d'un attribut est l'ensemble, fini ou infini, de ses valeurs possibles. Par exemple, l'attribut *numéroDeSécuritéSociale* a pour domaine l'ensemble des combinaisons de quinze chiffres et *nom* a pour domaine l'ensemble des combinaisons de lettres (une combinaison comme cette dernière est généralement appelée chaîne de caractères ou, plus simplement, chaîne).

3. Relation : Une relation est un sous-ensemble du produit cartésien de n domaines d'attributs (n > 0).

Un relation est représentée sous la forme d'un tableau à deux dimensions dans lequel les *n* attributs figurent les titres des *n* colonnes.

Voici un exemple de relation avec trois attributs :

numéro	nom	prénom
5	Durand	Caroline
1	Dubois	Jacques
12	Dupont	Lisa
3	Dubois	Rose-Marie

Personne(numéro : Entier, nom : Chaîne, prénom : Chaîne)

- Degré d ’une relation (arité) : son nombre d'attributs.
- Occurrence : est un élément de l'ensemble figuré par une relation. Autrement dit, une occurrence est une ligne du tableau.
- Cardinalité d'une relation : son nombre d'occurrences.
- Clé candidate : ensemble minimal des attributs de la relation dont les valeurs identifient de manière unique une occurrence.

REMARQUES

- La valeur d'une clé candidate est donc distincte pour toutes les occurrences.
- La notion de clé candidate est essentielle dans le modèle relationnel.
- Toute relation a au moins une clé candidate et peut en avoir plusieurs. Cela a pour conséquence qu'il ne peut jamais y avoir deux occurrences identiques au sein d'une relation : ces deux occurrences représenteraient en fait le même objet.
- Les clés candidates d'une relation n'ont pas forcément le même nombre d'attributs.
- Une clé candidate peut être formée d'un attribut arbitraire, utilisé à cette seule fin.
- Le contexte du domaine modélisé est essentiel pour déterminer les clés candidates d'une relation. Le contenu de la relation peut être un indice, mais il est parfois trompeur.

- Clé primaire d'une relation : est une de ses clés candidates.

- La notion de clé primaire est moins importante que celle de clé candidate dans le modèle relationnel.
- La clé primaire peut être choisie arbitrairement mais le contexte aide souvent à déterminer laquelle des clés candidates doit être considérée comme clé primaire.
- Pour signaler la clé primaire, ses attributs sont généralement soulignés.

- Clé étrangère d'une relation : est formée d'un ou plusieurs de ses attributs qui constituent une clé candidate dans une autre relation

- *Schéma de la relation r*, noté R

  = nom_rel (att:dom1, att2: dom2, …)
  - Convention : on omet les domaines : VILLE= (code, nom, adr)
- *Opération sur les relations :*
  - **projection** de la relation r de schéma R sur un s/ens d ’attributs S de R, notée $\pi_S(r)$ = la relation de schéma S obtenue en ne conservant que les colonnes de r qui correspondent aux attributs de S (et en supprimant les tuples dupliqués éventuellement)
- Exemple :projection de VILLE sur nom, adr

NOM	ADR
Dupont	Paris
Durant	Orsay
Dubois	Orsay

$r1 = \pi_{NOM, ADR}(r)$

ADR
Paris
Orsay

$r2 = \pi_{ADR}(r)$

– Equijointure de 2 rel. r et s, de schéma R et S, qui ont des attributs communs = relation notée r ⋈ s , ayant pour schéma R ∪ S (sans répétition), et dont les tuples sont obtenus en concaténant tout tuple de r avec tout tuple de s qui a les mêmes valeurs que lui sur les attributs communs de R et S.

– Ex. R=(nom, adr, num) et S = (num, code)

S

Num	code
2140	algo1
3213	BD
2555	BD

R

Nom	adr	num
dupond	paris	2140
durand	orsay	1128
dubois	orsay	3213

– alors r ⋈ s est :

Nom	adr	num	code
dupond	paris	2140	algo1
dubois	orsay	3213	BD

- **Schéma de Bd relationnelle** : ensemble de schémas de relations

- **Dépendance fonctionnelles (df)** :

  Soient : R, un schéma de relation, A, B, C des attributs et X, Y, Z des ensembles d ’attributs de R.

  => la relation r de schéma R vérifie la df X --> Y, si la connaissance des valeurs de X détermine celle des valeurs de Y (si 2 tuples ont même valeurs sur les attributs de X alors ils ont mêmes valeurs sur les attributs de Y)

  - Exemple : num-secu --> nom,
    - (reference, designation) --> quantite_commandee.

- **Propriétés des df** : soit R un schéma relationnel muni d ’un ensemble F de df.

  On note par XY la réunion des ensembles d ’attributs X et Y

  On note X-Y leur différence. Si X={A,B} et Y = {B,C} alors X-Y = {A}

  On appelle partie stricte Y d ’un ensemble X, tout sous-ensemble de cet ensemble non égal à lui. $Y \subset X$.

- **Implication de df** : la df X -->Y est impliquée par F si toute relation r de R qui vérifie F vérifie la df X --> Y. On note : $F \Rightarrow$ X --> Y

- **Fermeture de F** : (notée F^+) = ensemble des df impliquées par F.

- **Règles d ’Armstrong**

a) réflexivité : Si Y est contenu dans X, alors la df X --> Y est vérifiée. (df triviale)

b) Augmentation : pour tout Z inclus dans R, si la df X --> Y est vérifiée, alors la df XZ --> YZ l ’est aussi. Soit X --> Y ==> XZ --> YZ.

c) Transitivité : Si les df X --> Y et Y --> Z sont vérifiées, alors la df X --> Z l ’est aussi. {X --> Y et Y --> Z } ==> X --> Z

- **Intérêt de ces règles : cf théorème** :

Si la df X --> Y se déduit de F en appliquant les règles d ’Armstrong, alors X -->Y appartient à F^+.

Réciproquement, toute df X -- Y de F^+ se déduit de F par application de ces règles. Donc ces règles permettent de calculer F^+. Mais cela peut être fastidieux ==> règles supplémentaires (qui se déduisent des règles de base) :

d) additivité : {X --> Y et X --> Z } ==> X --> YZ.

e) Pseudo-transitivité : {X --> Y et WY --> Z } ==> XW --> Z.

f) Décomposition : X --> Y ==> X --> Z si Z est inclus dans Y.

- **Couverture minimale de F** = ensemble minimal de df représentant la même information que F, mais sans redondance.
- ***Calcul de la Couverture minimale de F** = ensemble G de df, tel que :*
  - *$G^+ = F^+$ (G implique les mêmes df que F)*
  - *tout membre droit d ’une df est réduit à un seul attribut.*
  - *Pour aucune df X--> A de G, on n’a : G - {X --> A} ==> G*
    - *(évite d ’avoir une df X --> A dont on peut se passer)*
  - *Pour aucune df X--> A de G, on n’a :*

    G ==> (G - {X --> A}) U Y --> A, avec Y partie stricte de X.

    ou pour aucune df X --> A de G, on n’a : G ==> Y --> A, où $Y \subset X$.

- En général, la couverture minimale n’est pas unique.
- Il existe des algorithmes pour la calculer.

- **Clé d ’un schéma de relation** : soit R = (A1, A2, …, An) un schéma de relation, F un ens. de df sur R et X un ensemble d’attributs de R.

  On dit que X est une clé de R muni de F si une des conditions suivantes est satisfaite :

  - *X --> R $\in$ F^+*
  - *F => X --> R*
  - *toute relation r sur R qui satisfait F vérifie X --> R*

- ***Fermeture d ’un ensemble d ’attributs X relativement à un ens. de df F*** *(noté X^+) = ens. des attributs A pour lesquels*

  la df X --> A est dans la fermeture de F. C ’est aussi l ’ens. des attr. qui prennent au plus une valeur quand celles des attr. de X sont fixées. (cf. algos de calcul de X^+)

- **conséquence** : vérifier qu ’un ens. d’attr. K est une clé de R muni de l ’ens. F de df revient à montrer que tous les attributs de R sont dans K^+. Mais seule la notion de clé minimale (appelée souvent clé ou clé candidate par opposition à superclé) est intéressante =>

- ***X est une clé minimale*** *(de R muni de F) ssi X est une clé et tout sous-ens. de X différent de X, n ’est pas une clé.*

Propriétés utiles

- ***P1*** *: tout attribut qui ne figure pas dans le membre droit d ’une df non triviale de F doit appartenir à toute clé de R.*

- ***P2*** *: si l ’ens. des attr. de R qui ne figurent pas en membre droit d ’une df non triviale de F est une clé, alors R possède une clé minimale unique formée par l ’ens. de ces attr.*

- ***P3*** *: un schéma de relation muni d ’une seule df possède une clé minimale unique.*

- **Décomposition d ’un schéma de relation** *:*

  $R=(A_1, A_2, \ldots, A_n)$ où les A_i sont des attributs simples, le remplacement de R par un ens de schémas de rel $R_1, R_2, \ldots R_p$ (p>=1) obtenus à partir de R par projection et tels que : $U_{i=1}^{i=p} R_i = R$

- ***Décomposition sans perte d ’information*** *(SPI) :*

  une décomposition de $R = (A_1, A_2, \ldots, A_n)$ est SPI si toutes les rel. r sur R considérées sont égales à la jointure des relations r_i $(1<=i<=p)$ obtenues par projection de r sur les schémas R_i.

- *Le théorème suivant (Ulman88) donne une CNS pour qu ’une décomposition soit SPI :*

  ***Théorème** : soit R=(X, Y, Z) où X, Y, Z sont des ensembles d ’attributs. Soit X --> Y dans F, alors la décomposition de R en S =(X,Y) et T =(X,Z) est SPI.*

  Réciproquement, si la décomposition de R en S et T est SPI alors X --> Y ou X --> Z appartient à F^+.

- ***Préservation des df par décomposition** : on dit que la décomposition de R en R_1, R_2, …, R_p préserve les df (ou sans perte de df, SPD) si la fermeture de la réunion des*

  $F_{i\ (1<=i<=p)}$ est égale à F^+. Soit $U\ (F_i)^+_{\ (1<=i<=p)} = F^+$.

- *Df élémentaire : R un schéma, X, Y, Z des ens d ’attr. ,*

  X --> Y est une df élémentaire, si c ’est une df et il n ’existe pas Z inclus dans X avec Z -> Y.
- *df directe : X -> Y est directe si : il n ’existe pas Z tel que*

  X -> Z et Z -> Y.

- ***Formes normales***

  La classification des relations en fonction de leur propriétés vis-à-vis des df est représentée par les formes normales.

- *Degré de normalité élevé => anomalies de màj réduites.*

- ***Relation en 1NF*** *: si tous ses attributs ont des valeurs atomiques (attributs non décomposables, 1 attribut => 1 valeur max)*

- ***Relation en 2NF*** *: si elle est en 1NF et aucun attribut non clé ne dépend d ’une partie de la clé (toutes les df sont élémentaires).*

- ***Relation en 3NF*** *: si elle est en 2NF et aucun attribut non clé ne dépend d ’un autre attribut non clé (toutes les df sont élémentaires directes).*

- *Ou bien une relation est en 3NF si tout attribut qui n ’appartient à aucune des clés minimales du schéma ne dépend que des clés du schéma et des ensembles d ’attributs qui le contiennent (cas de df triviales).*

- ***Remarques***

- **1.** un schéma est en 3NF si :
  - en cherchant ses clés minimales (algo), on déduit les attributs A qui n ’appartiennent à aucune clé minimale, puis on regarde les df X --> A de F^+ (où A est déjà déterminé),

    A non inclus dans X (cas de df triviale) et

    tester pour chacune d ’elle si X est une clé.

- **2.** Pour montrer qu’un schéma R **n ’est pas** est en 3NF, il suffit de donner une df X --> A de F^+ avec X non clé, A non inclus dans X et n ’appartenant à aucune clé.

- ***Relation en forme normale de Boyce-Codd (BCNF)*** *; si elle est en 3NF et tout attribut non-clé n 'est pas source de df vers une partie de la clé.*

- *Exemple : RECOLTE (product, annee, quantite, numvin)*

  On a : (product, annee) -> quantite

  (product, annee) -> numvin

  Mais on a aussi : numvin -> product

  d 'où la relation n 'est pas en BCNF.

- *On décompose en : RECOLTE(numvin, annee, quantite) et VIN(numvin, product) qui sont en BCNF.*

- ***Propriétés utiles :***

P4. Un schéma en 3NF qui n ’admet qu ’une clé minimale est en BCNF.

P5. Un schéma R muni d ’une seule df de la forme X -> Y,
avec X ∪ Y = R est en BCNF.

- **Algorithme de calcul de la fermeture d ’un ensemble d ’attributs X (c-à-d X^+):**

DEBUT
  X^+ := X
  REPETER
    aux := X^+ /**aux* est une variable auxiliaire*/
    POUR chaque df Y -> Z de F FAIRE
      SI Y est inclus dans X^+ ALORS X^+ := X^+ ∪ Z
    FINPOUR
  JUSQU ’A *aux* = X^+ ou X^+ = R
FIN

- ***Algorithme de recherche d ’une couverture minimale d ’un ensemble de df :***

X, Y, … = des ensembles d ’attributs,
A, B, C, … = des attributs somples

Données : *R=(A_1, A_2, …,A_n) un schéma de relation, F un ensemble de df sur R*
Résultat : *G, une couverture minimale de F*

(1) *G+ = F+*
(2) *Tout membre droit d ’une df de G est réduit à un seul attribut*
(3) *Pour aucune df X -> A de G on n ’a G - {X ->A } => X -> A*
(4) *Pour aucune df X -> A de G on n ’a G => Y -> A avec Y partie stricte de X*

ALGORITHME

- *On ordonne les df de F. Soit* $F = \{X_1 \rightarrow Y_1, X_2 \rightarrow Y_2, \ldots, X_n \rightarrow Y_n\}$
- ***Etape 1 :*** *on décompose les membres droits* Y_i *des df de F :*

  Pour i de 1 à m Faire

  Si $Y_i = A_1\ A_2 \ldots A_s$ *avec s > 1*

  Alors $F := F - \{X_i \rightarrow Y_i\} \cup \{X_i \rightarrow A_1, X_i \rightarrow A_2, \ldots, X_i \rightarrow A_s\}$

  On supposera par la suite que l 'on a : $F = \{X_1 \rightarrow A_1, X_2 \rightarrow A_2, \ldots, X_p \rightarrow A_p\}$
- ***Etape 2 :*** *on regarde si on peut enlever des df de F sans modifier sa fermeture*

  Pour i de 1 à p

  Si $\{F - \{Xi \rightarrow A_i\}\} \Rightarrow \{X_i \rightarrow A_i\}$ *Alors* $F := F - \{X_i \rightarrow A_i\}$

  Quitte à renuméroter les df, on suppose qu'à la fin de cette étape, on a :

  $F = \{X_1 \rightarrow A_1, X_2 \rightarrow A_2, \ldots, X_q \rightarrow A_q\}$
- ***Etape 3 :*** *On cherche à remplacer les membres gauches des df formés de plus d 'un attribut par des membres gauches ayant moins d 'attributs sans changer la fermeture de F*

  Pour i de 1 à q Faire

  Si Xi = B1 B2 … Br avec r > 1 Alors

  Pour j de 1 à r Faire

  Si $F \Rightarrow \{X_i - B_j\} \rightarrow A_i$ *Alors* $X_i := X_i - B_j$

- ***COMMENTAIRES***

- **Etape 1** *: propriétés d ’augmentation et de décomposition.*
  X -> AB est équivalent à X-> A et X -> B
- **Etape 2** *: on supprime des df superflues (redondantes)*
- **Etape 3** *: il suffit de remarquer que si on a F => ({X_i - B_j} -> A_i) alors les ensembles de df F et F ’ = {F - {X_i -> A_j} ∪ { {X_i-B_j} -> A_j} } sont équivalents.*

  En effet, par augmentation et décomposition, on a :

  la df {X_i - B_j} -> A_i implique la df X_i -> A_i (donc F ’ implique F)
- **Remarques :**
- *1. la couverture minimale n ’est pas unique (en général)*
- *2. on peut inverser l ’ordre des étapes 2 et 3, on obtient toujours une couverture minimale*
- *3. à l ’étape 2, pour tester si {F - {X_i -> A_i}} => {X_i -> A_i}, il suffit de tester si A_i appartient à la fermeture de X_i relativement aux df de*

  F - {X_i -> A_i}. De même à l ’étape 3, la condition F => {X_i - B_j} -> A_i peut se tester en regardant si A_i appartient à la fermeture de

  {X_i - B_j} relativement à F. (=> on peut utiliser l ’algo)

- ***ALGORITHME DE BERNSTEIN***

X, Y, … = ens d ’attributs, A, B, C, … = ens d ’attributs

Données : *R=(A_1, A_2, …,A_n) un schéma de relation,*

F un ensemble de df sur R

Résultat : *une décomposition de R muni de F en schémas de relations 3NF, SPI, SPD*

- ***Etape 1****. On remplace F pas sa couverture minimale (algo). On cherche les clés minimales de R et on teste si R est en 3NF.*

  Si oui, on s ’arrête, Si non on passe à l ’étape 2.

- ***Etape 2****. On regroupe les df X -> A_i ($1 \leq i \leq p$) ayant même membre gauche X. Pour chaque membre gauche X, on définit un schéma de relation contenant tous les attr. des df, soit*

  R_X=(X, A_1, A_2, …, A_p). Le schéma R_X est muni de l ’ensemble des df X -> Ai ($1 \leq i \leq p$).

- ***Etape 3.*** *Si aucun des schémas de R_X définis à l ’étape 2 ne contient de clé de R, alors on ajoute un schéma R_K =(K), où K est une clé minimale de R, muni d ’aucune df.*

- *REMARQUES :*

- *1. chacun des schémas R_i obtenus à l ’étape 2 et muni des df X -> A_i $(1 \leq i \leq p)$, soit encore de la df équivalente*
  X -> A_1 A_2…A_p, est bien en 3NF. De même le schéma R_K.

- *2. le schéma R_K sert à assurer que la décomposition est bien SPI.*

- *3. La décomposition est trivialement SPD puisque la réunion des df des nouveaux schémas est F.*

Les Systèmes de Gestion de Bases de Données (SGBD)

Langage SQL

Merci de me signaler les erreurs ...

SQL (Structured Query Langage) : standard -

4ème génération (SQL89, SQL2, SQL3)

--> traitement complet d ’une BD relationnelle

--> interactif ou par programmation

--> regroupe 3 langage :

* **DDL** (Data Description Langage): description des objets de la base (tables, …)

CREATE, ALTER, DROP

* **DML** (Manipulation) : --> Interrogation : SELECT

--> Manipulation : INSERT, UPDATE, DELETE

* **DCL** (Control) : contrôle d ’accès aux données : GRANT – REVOKE

* + 2 instructions : COMMIT (valider) et ROLLBACK (défaire)

On utilise comme exemple une BDR contenant 2 tables EMP et DEPT et une table DUAL d'un seul attribut. Elle est vide.

EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

DEPT (deptno, dname, loc)

DUAL table vide à une colonne

Traitement d ’une requête

- Analyse syntaxique : vérification de la disposition des mots dans la requête (mots-clé et autres)
- Optimisation : le SGBD génère des requêtes optimisées à partir de la requête de base. Il se sert pour cela de la connaissance de la structure des données, des index (s ’ils existent) et l ’organisation physique des données, et de statistiques d ’accès aux données.
- Exécution : le SGBD génère des plans d ’exécutions. Il en choit un et exécute la requête selon ce plan.

- Note : le SGBD tient compte du fait que d ’autres utilisateurs peuvent être en train d ’accéder aux mêmes données au même moment => contrôleur de concurrence.

SQL-DML

Interrogation : select

Extraction de toutes les lignes d ’une table

Select ***** **from** **nom_table ;**

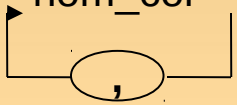
nom_commande *toutes les colonnes* *mot-clé* *nom de la table*

EX: extraire toutes les infos sur les départements : *Select * from dept*;

Info sur les employés : select * from emp;

Extraction d ’une ou plusieurs colonnes (projection)

Select nom_col from nom_table ;

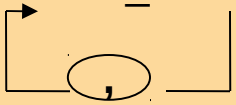


EX: Les noms des départements : *Select dname from dept;*

Noms et salaires des employés : *select ename, sal from emp;*

Eliminer les doublons éventuels

Select [**Distinct|ALL]** nom_col * from nom_table ; par défaut **ALL**



EX: les différents métiers (jobs) sans doublons : *Select distinct job from emp;*

Condition de recherche

- Select …. From … Where <condition>
- <condition> : suite de prédicats - Prédicat : comparaisons de 2 valeurs
- EX: Les employés du département 10 :

  *Select * from emp where deptno = 10;*

  – Les employés dont le salaire est > 3000 :

  *Select * from emp where sal > 3000;*

  – Les infos sur DALLAS :

  *Select * from dept where deptno = 'DALLAS';*

  Opérateurs : = , <>, < , > , <= et >=

  Les littéraux alphanumériques : entre apostrophes (') et sont comparés caractère par caractère (codage interne)

  Select … From …. Where [NOT] prédicat-1 AND|OR [NOT]prédicat-2 …;

  Ex : Les employés dont le job est ANALYST ou qui sont du département 10 :

  *Select * from emp where job='ANALYST' OR deptno = 10;*

  Employés dont le job est 'CLERK' et dont le salaire est > à 1000

  *Select * from emp where job = 'CLERK' AND sal > 1000;*

Opérateurs booléens

- 3 opérateurs booléens : NOT, AND et OR
- On peut les combiner dans une expression.
- ¨Priorité(NOT) > Priorité(AND) > Priorité(OR), sauf parenthésage.

Sélection de lignes dont la valeur d ’une colonne est comprise dans une suite de valeurs:

- Select … From …. Where nom_col [NOT] IN (val1, val2, …, val_n);
- EX : Les employés dont le chef a pour numéro soit 7902, soit 7839 :
- *Select * from emp where mgr IN (7902, 7839);*
- Les employés qui ne sont ni du département 20, ni du département 30 :
- *Select * from emp where deptno NOT IN (20, 30);*

Sélection de lignes dont la valeur d ’une colonne est comprise entre deux valeurs:

- Select * from emp where nom_col [NOT] BETWEEN val1 and val2;
- Les employés dont le salaire est compris entre 1500 et 2500 :
- *Select * from emp where sal between 1500 and 2500; /* bornes incluses */*

Opérateurs arithmétiques

- +, -, *, / : combinés pour faire des expressions.
- EX : Afficher les employés avec leur salaire total (avec la commission quand elle existe).
- *Select ename, sal, comm, sal+comm where comm is not null;*

Tri des résultats

- Les résultats d ’une sélection peuvent être triés, sur une ou plusieurs colonnes, par ordre croissant (par défaut) ou décroissant.
- Select * from emp where ….. ORDER BY nom_col [ASC|DESC] ;

- Employés dont le salaire est < 1500, triés par ordre alphabétique croissant :

*Select * from emp where sal < 1500 ORDER BY ename;*

- Note : Il est possible d ’effectuer des tris sans mentionner les noms de colonnes mais leurs positions relatives dans le SELECT :
- Liste des employés triés par numéros d ’employé croissant, et sur le numéro du chef décroissant:

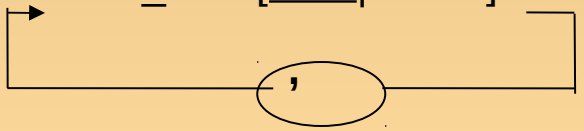
Select empno, ename, job, mgr from emp ORDER BY 1, 4 DESC;

Opérateurs arithmétiques

- +, -, *, / : combinés pour faire des expressions.
- EX : Afficher les employés avec leur salaire total (avec la commission quand elle existe).
- *Select ename, sal, comm, sal+comm where comm is not null;*

Tri des résultats

- Les résultats d ’une sélection peuvent être triés, sur une ou plusieurs colonnes, par ordre croissant (par défaut) ou décroissant.
- Select * from emp where ….. ORDER BY nom_col [ASC|DESC] ;



- Employés dont le salaire est < 1500, triés par ordre alphabétique croissant :

  *Select * from emp where sal < 1500 ORDER BY ename;*
- Note : Il est possible d ’effectuer des tris sans mentionner les noms de colonnes mais leurs positions relatives dans le SELECT :
- Liste des employés triés par numéros d ’employé croissant, et sur le numéro du chef décroissant:

  Select empno, ename, job, mgr from emp ORDER BY 1, 4 DESC;

La Jointure

- Association de lignes de plusieurs tables en fonction d ’un critère (de jointure).

  Select … From nom_tab [nom_alias] Where <critère de jointure>;

- Afficher les noms des employés (table emp) avec leur nom de département (table dept) :

  Select ename, dname from emp, dept where emp.deptno = dept.deptno ;

- Rmp : le nom de colonne étant identique => on le préfixe par le nom de la table.

Auto-jointure (jointure de lignes différentes de la même table)

- Lister les employés avec leur manager : le critère d ’auto-jointure est réalisé en associant EMPNO et MGR (qui est aussi un numéro d ’employé - du chef -):
- Pour effectuer l ’auto-jointure, il convient de citer 2 fois la même table en utilisant un alias, car on traite 2 tables identiques, les mêmes noms de colonnes apparaissent dans les deux tables.
- *Select emp.empno, emp.ename, empbis.empno, empbis.ename*

  From emp, emp empbis

  Where emp.mgr = empbis.empno;

Sous-interrogation

- Select … From … Where nom_col <opérateur> (Select … From … Where…);

- EX: Afficher les employés qui sont dans le même département : recherche du département de ALLEN puis connaissant son numéro (30), on cherche les employés qui y travaillent.

*Select * From emp Where*
deptno = (Select deptno from emp where ename = 'ALLEN');

Sous-interrogation rapportant plusieurs lignes

Le résultat d 'une sous-interrogation peut comporter plusieurs lignes. Dans ce cas les opérateurs =, <, > … ne conviennent plus. L 'égalité sera traitée par l 'opérateur IN (un '=' par rapport à une suite de valeurs).

- Les inégalités seront traitées par l 'opérateur ANY et ALL.

- Exemple : Les employés travaillant dans le même département que l 'un des employés dépendant du président.

*Select * from emp*
Where deptno in (select deptno from emp
Where mgr = (Select empno from emp
Where job=' PRESIDENT '));

Opérateur ANY : Le résultat de la comparaison est VRAI, s ’il l ’est pour au au moins un élément de l ’ensemble (sous-interrogation)

EX: Afficher les employés ayant un salaire supérieur à celui de l ’un des employés travaillant dans le département 10.

```
Select * from emp
        where sal > ANY (select sal from emp
                                where deptno=10);
```

Opérateur ALL : Le résultat de la comparaison est VRAI, s ’il l ’est pour tous les éléments de l ’ensemble (sous-interrogation)

EX: Afficher les employés ayant un salaire supérieur à celui des employés travaillant dans le département 20.

```
Select * From emp
        where sal > ALL (select sal from emp
                                where deptno=20);
```

Extraction de lignes si le résultat de la sous-interrogation comporte au moins une ligne

Select … From …. Where [NOT] EXISTS
(Select … From … Where …);

Ex1: lister les employés s 'il y en un parmi eux qui a une commission > 1000;

*Select * from emp where EXISTS*
*(select * from emp where comm > 1000);*

Ex2 : lister les employés s 'il n 'y a aucun parmi eux qui a une commission > 1000

*Select * from emp where NOT EXISTS*
*(select * from emp where comm > 1000);*

Les traitements de groupe

Select [AVG|SUM|MIN|MAX|COUNT] ([DISTINCT|ALL] nom_col)
Where condition;
Le résultat sera affiché sur une seule ligne.

Ex: Afficher le total des salaires : *Select sum(sal) from emp;*
Afficher le total et la moyenne des commissions :
Select sum(comm), avg(comm) from emp;

Afficher le nombre de commissions à NULL :
Select count(comm) from emp;

Rmq1 : la valeur NULL n 'entre pas dans les différents calculs.

Ex : la moyenne de la colonne COMM est calculée en cumulant le nombre de commissions divisée par le nombre de valeurs non NULL.

Rmq2 : Impossible d 'utiliser à la fois une projection et une fonction de groupe : *Select ename, sum(sal) from emp;* SERA REJETEE.

Rmq3 : COUNT (*) : nombre de lignes non NULL.

Regroupement du résultat d ’une sélection

Les lignes provenant d ’une sélection peuvent être regroupées en fonction d ’une valeur commune dans une ou plusieurs colonnes :

Select [AVG| …|COUNT] ([distint|all] from nom_table where <condition>
GROUP BY [nom_col1 | nom_col2| ….];

Rmq: Il n ’est pas nécessaire de réaliser un tri avant le regroupement : il est effectué automatiquement par GROUP BY.

Ex: Calculer le total des salaires par département :

Select deptno, sum(sal) from emp group by deptno;

Sous-interrogation de groupe

Select …. From nom_tab [alias] Where nom_col <operateur>
(select … from … where …)

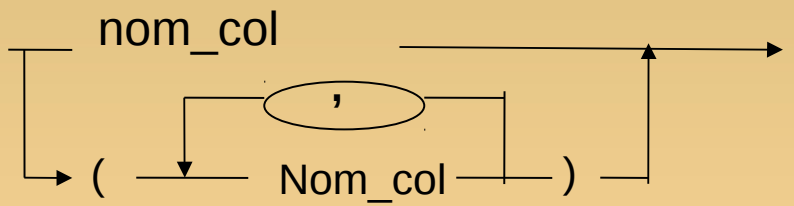
Ex: Afficher les employés ayant le plus grand salaire par département :

Select ename, sal, deptno, from emp e1
Where sal = (Select max(sal) from emp
where e1.deptno= emp.deptno);

Rmq : pour lever l'ambigüté sur le nom de colonne (appartenant soit à l 'interrogation pple, soit à la sous-interrog), on utilise un alias.

Sous-interrogation de groupe (II)

Select …. From nom_tab Where



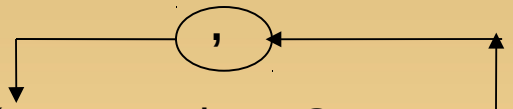
<operateur> (select … from … where …);

Ex: Afficher les employés ayant le plus grand salaire par département :

Select ename, sal, deptno, from emp
Where (sal, deptno) IN (
Select max(sal), deptno from emp
GROUP BY deptno
);

Création d ’une table à partir d ’une autre table

CREATE TABLE nom_tab AS SELECT …. FROM …. WHERE …..;
Ou bien



CREATE TABLE nom_tab (nom_col NOT NULL) ⟶ AS SELECT …….;

Ex: Create table dept30 (name, hdate, manager)
AS SELECT ename, hiredate, mgr From emp Where deptno = 30;

Rmq : Si aucune colonne n’est spécifiée, les noms de colonnes sont ceux de la table d ’origine.

- Si des colonnes sont spécifiées, leur nombre doit être le même que celui de l ’interrogation (après AS SELECT)
- Les types des colonnes sont ceux des colonnes de la table d ’origine
- Cette commande correspond à une création et une insertion de valeurs.

Rmq : la syntaxe peut différer d'un SGBD à un autre (COPY sous postgresql, ..)

Les types de données

Alphanumériques : CHAR(n) chaîne fixe
VARCHAR (n) : chaîne variable (<=n car), table de n positions.
VARCHAR2(n) : vraie longueur variable
Numérique : NUMBER [* | n], INT, INTEGER, SMALLINT
NUMBER(n, nb_decimales), DECIMAL, FLOAT
Type DATE : date depuis le 01/01/4712 avant J.C au 31/12/4711 après J.C.
Par défaut : ‘dd-mon-yy’

Create table jours (d date);

Insert into jours values (‘06-nov-97’);

Oracle permet des conversions : TO_DATE

Insert Into jours values (to_date(‘2000-01-01’, ‘yyyy-mm-dd’));
=> stockée : ‘ 01-jav-00 ’

Sélection de groupes de lignes en fonction d ’un critère

Select [MIN ……. COUNT] [distinct|all] [(nom_col)|*] From nom_tab
Where <condition> GROUP BY [nom-c1|nom_c2|….]
HAVING [NOT] prédicat1 AND|OR] [NOT] prédicat2 …..;

Rmq. La clause HAVING pour les groupes de lignes est similaire à la clause WHERE pour les lignes.

Ex: Afficher les départements dont le cumul des commissions est > 0 :

```
Select deptno, sum(comm) from emp
        GROUP BY deptno
       HAVING sum(comm) > 0;
```

- Afficher le département ayant le plus grand cumul de salaire :

```
Select deptno, sum(sal) from emp
        GROUP BY deptno
        HAVING sum(sal) > (Select MAX(SUM(sal)) from emp
                                GROUP BY deptno);
```

Opérateurs sur plusieurs tables

Opérateurs algébriques : union, inter ou intersect et minus

Exemple : 2 tables de même structure et de contenus dfférents : depot1 et deptot2:

Structure : numprod, design, conditionnem, stock, rayon, pu, tva

UNION : lignes de une ou plusieurs tables (sans les doublons)

Ex : Lister les produits en stocks dans les 2 dépôts :

```
Select numpro, design, pu, from depot1
UNION
Select numpro, design, pu, from depot2;
```

Différence (MINUS) : lignes existant dans une table mais pas dans une autre

Ex : Lister les produits en stocks dans le dépôt1 uniquement :

```
Select numpro, design, pu, from depot1
MINUS
Select numpro, design, pu, from depot2;
```

Lignes communes à plusieurs tables

INTERSECT : les doublons sont éliminés.

Select …. From …. Where ….
INTERSECT
Select …. From …. Where ….;

Ex: Afficher les produits identiques qui sont en stock dans les 2 dépôts :

Select numpro, design, pu From depot1
INTERSECT
Select numpro, design, pu from depot2;

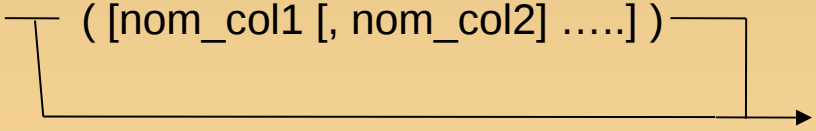
Rmq : Les noms de colonnes du résultats sont les noms de colonnes du 1er SELECT.

SQL-DML

mise à jour : update, delete, insert

On travaille sur les tables EMP et DEPT.

Ces commandes permettent d ’insérer des valeurs dans des tables, de supprimer des valeurs ou de mettre à jour des valeurs.
INSERT : insertion d ’une ligne dans une table

INSERT INTO nom_tab 

VALUES (val1 [,val2] ….) ;

Ex1: Insérer un nouveau département : 50, ‘EDUCATION’, ‘MIAMI’ :

Insert INTO dept VALUES (50, ‘EDUCATION’ , ‘MIAMI’);

On n ’a pas à citer les colonnes quand on insère des valeurs pour toutes les colonnes !

Ex2:Insérer un employé en ne connaissant que le numéro, le nom, le job et le numéro de département :

Insert into EMP (empno, ename, job, deptno)
VALUES (7950, ‘ JOHN ’, ‘ TRAINER ’, 50);

Suppression : DELETE

DELETE FROM nom_tab [Where condition];

Supprimer toutes les lignes d 'une table : BONUS :

Delete from bonus;

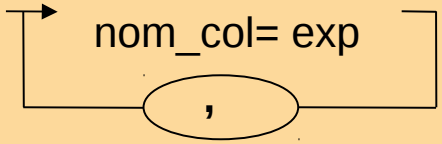
Supprimer les employés ayant une commission non NULL :

Delete from emp where comm is not null;

Rmq : L 'utilisateur doit être propriétaire de la table ou avoir les droit DELETE sur celle-ci.

Si la clause WHERE est absente, toutes les lignes seront supprimées.

MISE A JOUR : UPDATE :

Update nom_tab SET 

Where condition;

Ex: Remplacer dans le département 30, la localité 'CHICAGO' par 'LOS ANGELES' :

UPDATE dept SET loc = 'LOS ANGELES' Where deptno=30;

DELETE (suite)

Augmenter la commission de 10% pour tous les employés :

*Update emp SET comm = comm * 1.1;*

Rmqs :

- L ’utilisateur qui lance la commande doit être propriétaire de la table ou avoir reçu le droit UPDATE sur la table

- Le verbe SET indique les colonnes à modifier. Seules les colonnes spécifiées seront modifiées.

- Si la condition WHERE est absente, toutes les lignes spécifiées dans SET seront modifiées.

- L ’effacement d ’une colonne est réalisé en utilisant l ’option NULL : SET nom_col = NULL;

Validation - Annulation des transactions

- Validation des modifications effectuées :

COMMIT [work release]; /* *work release* par compatibilité avec les versions antérieures */

- Annulation des effects d ’une transaction :

ROLLBACK [work release]; /* *work release* par compatibilité avec les versions antérieures */

Rmq : En général, avec SQL interactif, chaque ordre est une transaction. Donc les modifications éventuelles se répercutent automatiquement (Commit implicite pour l ’utilisateur)

- COMMIT et ROLLBACK sont très utiles quand on utilise un langage procédural et/ou une interface avec un langage de programmation.

Expressions et Fonctions

Une expression : combinaison de variables (contenu d ’une colonne), de constantes et d ’autres expressions à l ’aide des opérateurs : +, -, *, /

Une fonction : routine ayant des arguments et ramenant un résultat (un argument peut être une expression)

Il y a 3 types d ’expressions : arithmétiques, chaînes de caractères et date.

A chaque type ==> il y a des opérateurs spécifiques.

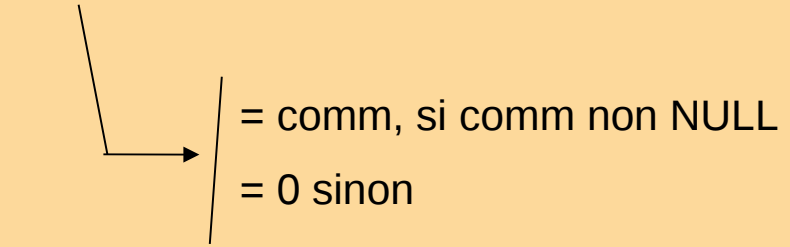
Exemple de fonctions : NVL : NULL VALUE

permet de remplacer une valeur NULL par une valeur significative.

Syntaxe : NVL(exp1, exp2) :

renvoie la valeur de exp1, si exp1 non NULL et de exp2 sinon.

Ex: Select ename, sal, comm, sal+NVL(comm, 0) from emp;



Rmq : sous Postgresql, COALESC(liste)
vaut la première valeur NOT NULL.

Autres fonctions : il existe d'autres fonctions :

CEIL, ROUND, CONCAT, ...

Insertion de lignes dans une table à partir d ’une autre table

INSERT INTO nom_tab (nom_col [, nom_col] ...)
SELECT …… FROM ……. WHERE ……..;

Ex: Insérer dans une table BONUS (ename, job, sal, comm), à partir de la table emp, les employés qui sont SALESMAN et dont la commission est supérieure à 25% du salaire :

Insert Into BONUS
Select ename, job, sal, comm From emp
*Where job=‘SALESMAN’ and COMM > 0.25*sal;*

SQL-LDD (Description)

CREATION :

CREATE TABLE nom_tab (<definition_colonne> [, <definition_colonne>] …);

Où <definition_colonne> est :

nom-col type [DEFAULT val_defaut] [NOT NULL] [UNIQUE] [autre]);

Où autre peut être :

- … REFERENCES nom_tab [(nom_col [, nom_col] …)] …;
- … CHECK condition;

Contraintes :

NOT NULL : si on ne spécifie rien pour la valeur d ’un attribut, il prend la valeur NULL (indéterminée). Si la clause NOT NULL est présente, on doit spécifier une valeur non nulle pour cet attribut.

DEFAULT : on peut spécifier une valeur par défaut pour un attribut à la création de la table (cette valeur doit être du type de l’attribut).

UNIQUE : spécifie qu’un attribut (ou groupe) est une clé candidate. 1 seul tuple correspond à une valeur de cette clé. Il est conseillé de spécifier NOT NULL pour un attribut déclaré UNIQUE (qui n ’est pas clé primaire).

Exemple : Create table fournisseurs
(… nom_fou char(25) NOT NULL UNIQUE …);

CHECK : spécifie une contrainte qui doit être vérifiée à tout moment par les tuples concernés.

Exemple : (Create table ….
(cli_type char(16) DEFAULT ‘PARTICULIER’
CHEK (cli_type IN (‘PARTICULIER’, ‘ PME ’, ‘ AUTRE ’)

La quantité livrée doit être inférieure ou égale à la quantité commandée.

… qte_liv integer default 0 CHECK (qte_liv <= qtç_com….)

Avec NOT NULL : …… CHECK (nom_fou IS NOT NULL) …..

Intervalle : …. CHECK val_prix BETWEEN 10 AND 40

Contrainte d 'entité : **PRIMARY KEY**

1- Create table client (
numcli char(5) not null primary key, …..);

2- Create table lign_comm (ref char(5) not null, design char(10) not null, primary key (ref, design), …..);

CI référentielle : => gestion correcte des modif de la clé étrangère dans la table qui référence et de la clé primaire dans la table référencée.

Soient les relations :

client(<u>numcl</u>, ….) et commande(<u>numcom</u>, …numcl,..)

si on ajoute une commande => il faut un client associé.

3 solutions pour gérer les CI référentielles :

1. Fonction spécifique écrite par le programmeur : solution souple (on y met ce qu ’on veut) mais fastidieuse (une fonction dans chaque application).

2. Vérification assurée par le SGBD automatiquement : la contrainte est déclarée explicitement une fois pour toutes (valable pour tout programme et utilisateur).
=> adoptée par la norme SQL-92 (beaucoup de SGBD).

3. « trigger » approprié déclenché automatiquement. = une fonction compilée, écrite une fois et exécutée automatiquement chaque fois qu’une action particulière est effectuée.

Mise en œuvre de la solution 2 (intégrité référentielle déclarative).

```
Create table client (num_cl char(5) not null primary key, ……);
Create table commande(numcomm integer not null primary key, ….
        Num_com char(5)  not null REFERENCES client, ….);
```

Si la clé étrangère = plusieurs attributs => utiliser FOREIGN KEY.

```
… FOREIGN KEY (atr1_cle, attr2_cle) REFERENCES (cle_prim1).
```

On ajoute la clause suivante : …. Num_com references client

```
[ON UPDATE {option}]
[ON DELETE {option}]
```

où option peut être l ’une des options suivantes :

Point de vue de la table référencée

Si on modifie la valeur d ’une clé primaire référencée, SQL permet de réaliser automatiquement certaines opérations :

NO ACTION : équivalent à ne pas mettre de clause ON UPDATE ou ON DELETE => refus de modifier ou supprimer une clé primaire.

ON UPDATE [ON DELETE] CASCADE : si UPDATE d ’une clé primaire référencée => les modif sont répercutées sur les relations qui référencent. SI DELETE => suppression des clés qui référencent.

SET NULL : si une clé primaire référencée est modifiée => on les met à NULL là où elles sont référencées (condition : les parties de la clé étrangères doivent permettre les valeurs NULL).

SET DEFAULT : si la clé primaire est modifiée => la valeur de la clé étrangère qui référence est mise à une valeur par défaut (définie au préalable)

Exemple :

```
…. deptno REFERENCES dept
                ON UPDATE CASCADE
                ON DELETE SET NULL;
```

Modification de la structure d ’une table

ALTER TABLE table_name ADD column_name datatype ;

ALTER TABLE table_name DROP COLUMN column_name ;

ALTER TABLE table_name MODIFY column_name datatype ;

EX : ALTER TABLE emp ADD (diplôme char(20));

ALTER TABLE EMP (sal numer (8,2));

Suppression d ’une table

DROP TABLE nom-tab;

Ex : Drop table bonus;

SQL - LCD

Contrôle des accès à la base et aux objets

CREATION DES UTILISATEURS AVEC PRIVILEGES

GRANT [CONNECT|RESOURCE|DBA] TO nom-user

IDENTIFIED BY mot-passe;

EX: Le DBA créé l ’utilisateur USER1 avec les privilèges de connexion et de création d ’objets :

Grant connect, resource to USER1 identified by lambda;

Reprise de privilèges

REVOKE [CONNECT|RESOURCE|DBA] FROM nom-user ;

EX : Revoke resource from user1;

Attribution de droits sur les objets

Les objets doivent être la propriétés de l ’utilisateur ou qu ’il a reçu les droits nécessaires :

GRANT [SELECT|INSERT|DELETE|UPDATE|INDEX|ALL]

ou bien

GRANT UPDATE (nom-col, nom-col, …)

ON nom-tab1, nom-tab2, … TO [U1, U2, ...|PUBLIC]

WITH GRANT OPTION;

Où :

INDEX : donne la possibilité de créer des index. Exemple :

CREATE [unique] INDEX ind ON nom-tab (nom-col1 [asc|desc], nom-col2 [asc|desc], …);

/* pour améliorer les performances lors de la manipulation */

ALL : tous les droits

L ’option : «WITH CHECK OPTION» signifie que celui a qui l ’on accorde les droits spécifiés peut lui-même les transmettre à d ’autres.

EX : Grant select, update on emp to user1 with check option;

Annulation des droits

REVOKE [SELECT| …..] ON nom-tab, nom-tab2, ..FROM [U1, U2, …| PUBLIC]

EX : Revoke update on cours to user2;

Création d ’un synonyme pour une table

CREATE [PUBLIC] SYNONYM nom-syn FOR nom-table;

EX: Un utilisateur se crée un synonyme pour sys.emp :

Create synonym l-emp For SYS.emp;

L ’utilisateur SYSTEM (DBA) crée un synonyme accessible à tout le monde pour sa table emp :

EX: Create public synonym p-emp for SYS.emp;

Les Systèmes de Gestion de Bases de Données (SGBD)

Gestion des transactions

Gestion des transactions

Reprise après panne - fiabilité

On entend par fiabilité (reliability) d ’un système sa capacité à surmonter les erreurs et anomalies qui y surviennent.

- Principales erreurs ou dysfonctionnements :
  - erreurs de programmation des transactions : locales
  - erreurs systèmes (défaillances) : globales
  - erreurs de médias
- => induisent le fonctionnement incorrect de la base ou sa destruction.

Objectifs des mécanismes qui assurent la fiabilité

- Eviter les incohérences de la base
- Rétablir la cohérence de la base suite à un problème

- **Erreurs de médias** (disques) : destruction de parties de la base => solution : avoir des copies d ’archives sur supports distincts et/ou BD dupliquées
- **Erreurs locales** (transactions) : résolues à l ’aide de contraintes d ’intégrité et/ou triggers, …
- **Erreurs systèmes** : suite à une panne (principalement coupure de courant)

  ==> perte du contenu des buffers de la mémoire centrale (volatile), donc perte des information des transactions actives

Solution aux défaillances système

- Avoir des duplicatas des données (invisibles à l ’utilisateur)

  ==> le système utilise un journal (fichiers log)

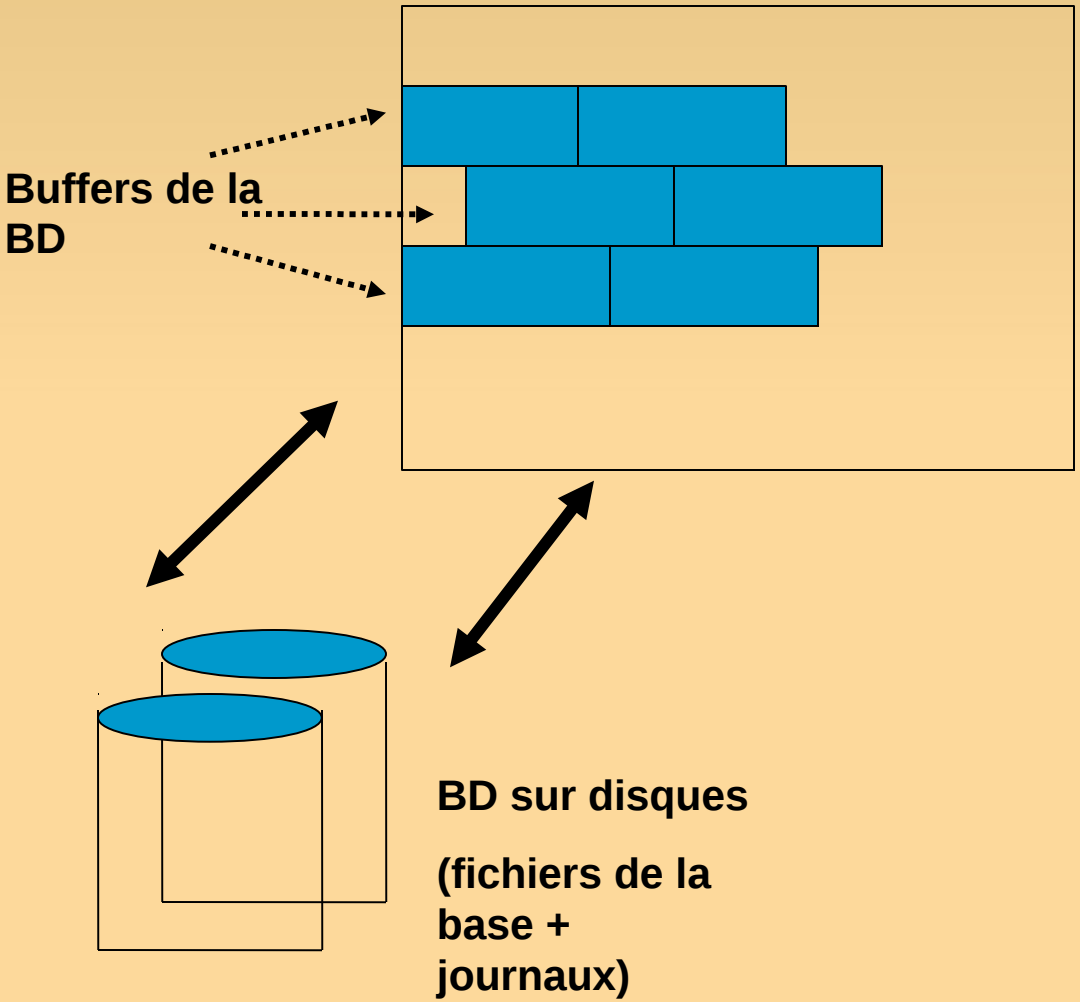
- Pour restaurer la base dans un état cohérent : déroulement d *’une procédure de reprise*

- Un programme = une séquence de transactions
- Une transaction = une séquence d ’actions (lire, écrire, calculs en MC), Mais pour la BD, les opérations importantes sont lire/écrire.

- Une transaction :
  - début transaction
    - accès à la base (lire/écrire)
    - calculs en MC
  - fin transaction (COMMIT ou ROLLBACK)

- COMMIT : fin de la transaction avec succès (les modifications qu ’elle a effectuées deviennent permanentes dans la base)
- ROLLBACK : fin de la transaction avec échec (la base revient à l ’état où elle était avant le début de la transaction)
- Rmq : le gestionnaire de recouvrement doit faire en sorte que la propriété d ’atomicité de la transaction soit maintenue, suite à une panne (comme d ’annuler les effets des actions de transactions non terminées par COMMIT)

- Une BD = ens. de fichiers sur supports stables (disques)
- Le SGBD exploite la base en travaillant sur une partie de cette base en MC (dans des buffers)



Journalisation

- Écriture sur support stable du déroulement de l ’activité des transactions = pour que le SGBD puisse respecter l ’atomicité des transactions

- Il y a 3 types de journalisation :
  - UNDO (défaire)
  - REDO (refaire)
  - UNDO/REDO (défaire/refaire)

- Mécanisme de reprise après panne (recovery) : permet de remettre la BD dans un état cohérent suite à une panne (défaillance système, perte du contenu des buffers en MC)

- Une partie du journal est localisée dans des buffers en MC, et sont écrits périodiquement (ou forcés) sur le log sur disque.

- Suite à une panne, le processus de recouvrement consiste à :
  - annuler les actions effectuées par les transactions (UNDO)
  - refaire les actions effectués par les transactions (REDO)
  - refaire certaines actions et en annuler d ’autres (UNDO/REDO)

Différents types d ’enregistrements dans le log

- <start T> : début transaction
- <Commit T> : validation de la transaction
- <Abort T> : annulation de la transaction
- <T, X, old_v, new_v> : la transaction T a mis à jour la donnée X avec la nouvelle valeur new_v (l ’ancienne valeur étant old_v)

- Rmq : <T, X, old_v, new_v> est généré par un write (l ’écriture de l ’enregistrement ne se fait pas forcément sur disque)

La gestion des transactions comporte, entre autres, la reprise après panne et le contrôle de concurrence.

Reprise après panne : restaure la base de donnée dans un état considéré correct, suite à une panne.

Idée sous-jacente : la redondance (transparente à l ’utilisateur). Donc le moyen pour restaurer une base consiste à garantir que toute info qu ’elle contient peut être reconstruite à partie d ’autres infos mémorisées quelque part dans le système.

Transaction: Notion fondamentale dans un SGBD = unité logique de travail. Est considérée comme une unité atomique.

```
Exemple1 : Insert into SP (s : ’S5 ’, P : ’P1 ’, QTY: 1000);
            If <erreur> Go To undo;
           Update SP set QTY := QTY + 500 Where P := ‘ P1 ’;
           If <erreur> GO TO undo;
           Commit transaction;
           Go To fin;
  undo :   rollback transaction
  fin :    return;
```

Une transaction : plusieurs opérations élémentaires.

Si elles sont toutes exécutées : transforment un état cohérent de la base en un autre état cohérent (atomicité).

L ’idéal : toutes les actions s ’exécutent sans problèmes. Mais :

défaillances systèmes, débordements, ….

=> le gestionnaire de transactions doit garantir que si certaines actions (mises à jour) ne sont pas exécutées, alors il est capable de restaurer un état cohérent de la base (ex. annuler les màj effectuées). Donc soit la transaction s ’exécute entièrement, soit pas du tout => une séquence d ’opérations non atomique apparaît comme atomique au niveau externe.

Cette atomicité est fournie par le gestionnaire de transaction (notamment avec les opérations COMMIT et ROLLBACK)

COMMIT : signale la fin d ’une transaction avec succès. Les màj effectuées peuvent maintenant devenir permanentes.

ROLLBACK : signale la fin anormale de la transaction. La BD pourrait être dans un état incohérent et les màj effectuées doivent être défaites (annulées) pour revenir au point de départ.

Comment annuler une màj ?

Grâce à un journal (fichier log) localisé sur support stable, et géré par le système.Il contient toutes les màj (valeurs avant et valeurs après modif, notamment) .Pour annuler une màj particulière, le système utilise l ’entrée correspondante du journal pour retrouver la valeur avant màj.

En pratique, le journal = 2 parties. Une partie active sur disque (accès direct) et une partie archive sur bande (accès séquentiel). Quand la partie active est pleine, son contenu est transférée vers l ’archive.

Autre pb : garantir que chaque opération élémentaire est, elle-même, atomique. Crucial, notamment pour une BD relationnelle où les opérations portent sur plusieurs lignes d ’une table en même temps. Il ne faudrait pas qu ’une opération échoue au milieu de la màj d ’une table.

Reprise d ’une transaction après panne :

Une transaction commence par BEGIN TR (ou équivalent) et se termine par COMMIT (si succès) ou ROLLBACK (si échec).

Le COMMIT établit un point de commit (syncpoint). Il correcspond à la terminaison d ’une unité logique de travail, à un point où la BD est (ou devrait être) cohérente.

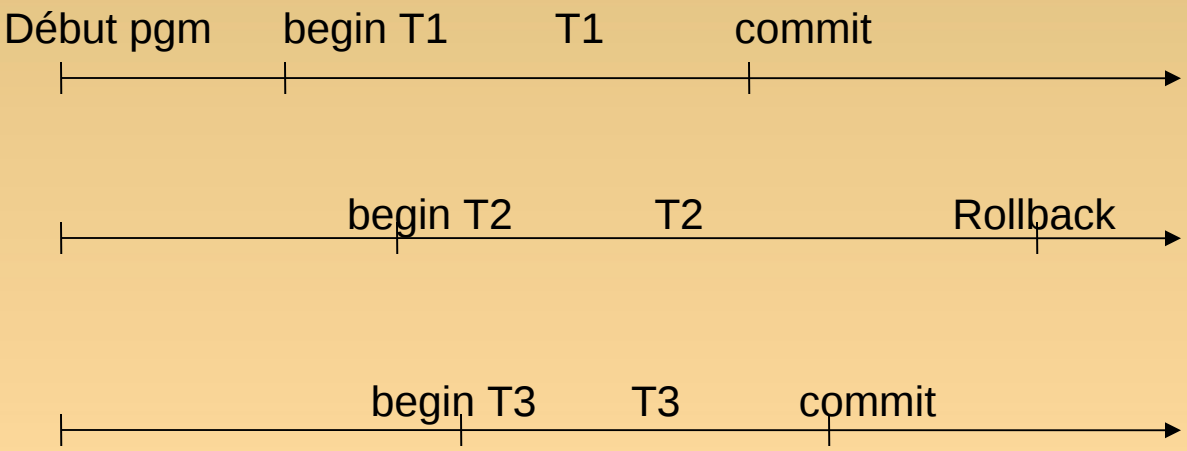
Le ROLLBACK remet la BD dans l ’état où elle était avant le BEGIN TR (retour au point de commit de la transaction précédente).

Lorsqu ’un point de commit est établi :

- toutes les màj terminées par COMMIT effectuées depuis le dernier point de commit sont validées (deviennent permanentes). Avant ce point, les màj sont assimilées à des tentatives, car elles pourraient être annulées.
- Toutes les variables de positionnement (références à des curseurs, par exemple) et tous les verrous sont effacés.

Exécution d ’un programme :

En général, un programme est une séquence de plusieurs transactions qui s ’exécutent les unes après les autres.



Remarque : Dans l ’exemple1, des tests explicites sont inclus pour tenir compte des erreurs. Mais d ’autres erreurs non prévues peuvent survenir => le système exécute des ROLLBACK implicites pour ces programmes.

La transaction est également une unité de reprise :

Si la transaction se termine par COMMIT (succès) le système garantit que les màj deviennent permanentes, même si une défaillance survient après le COMMIT. Cela est possible car, au COMMIT, les màj pourraient avoir été effectuées sur un tampon en MC, et si une défaillance se produit juste à ce moment là, la procédure de redémarrage du système est capable de positionner ces màj en examinant les entrées du journal (LOG). Il s ’ensuit que le contenu du journal doit être écrit physiquement sur support stable avant de terminer l ’opération de COMMIT. Cette règle est appelée WAL (Write-Ahead Log rule) ou écriture en avant dans le journal.

Propriétés ACID des transactions : une transaction doit posséder les 4 propriétés suivantes Atomicité, Cohérence, Isolation et Durabilité.

Atomicité: toutes les opérations d ’une transaction sont exécttées ou aucune ne l ’est (principe du tout ou rien).

Cohérence : une transaction préserve la cohérence de la BD. Elle transforme un état cohérent de la BD en un autre état cohérent.

Isolation : une transaction s ’exécute isolément des autres. Même si plusieurs transactions s ’exécutent en concurrence, les màj de chacune ne deviennent visibles aux autres transactions qu ’après le COMMIT.

Durabilité : une fois qu ’une transaction est validée (COMMIT), ses màj deviennent permanentes dans la base, même si ensuite une panne survient.

Reprise du système:

- Défaillance locale (transaction particulière) : discutée précédemment. N ’affecte que la transaction dans laquelle la défaillance s ’est produite.
- Défaillance globale : affecte toutes les transactions en cours d ’exécution au moment de la défaillance => nombreuses conséquences au niveau du système. Il en existe 2 types :
  - défaillances système, dites douces : pannes de courant, …
  - défaillances des supports, dites dures : disques endommagés, fichiers perdus, ...

On s ’occupe des **défaillances douces**, les autres défaillances nécessitent d ’avoir des copies de sauvegarde, un journal, des utilitaires de sauvegarde,....

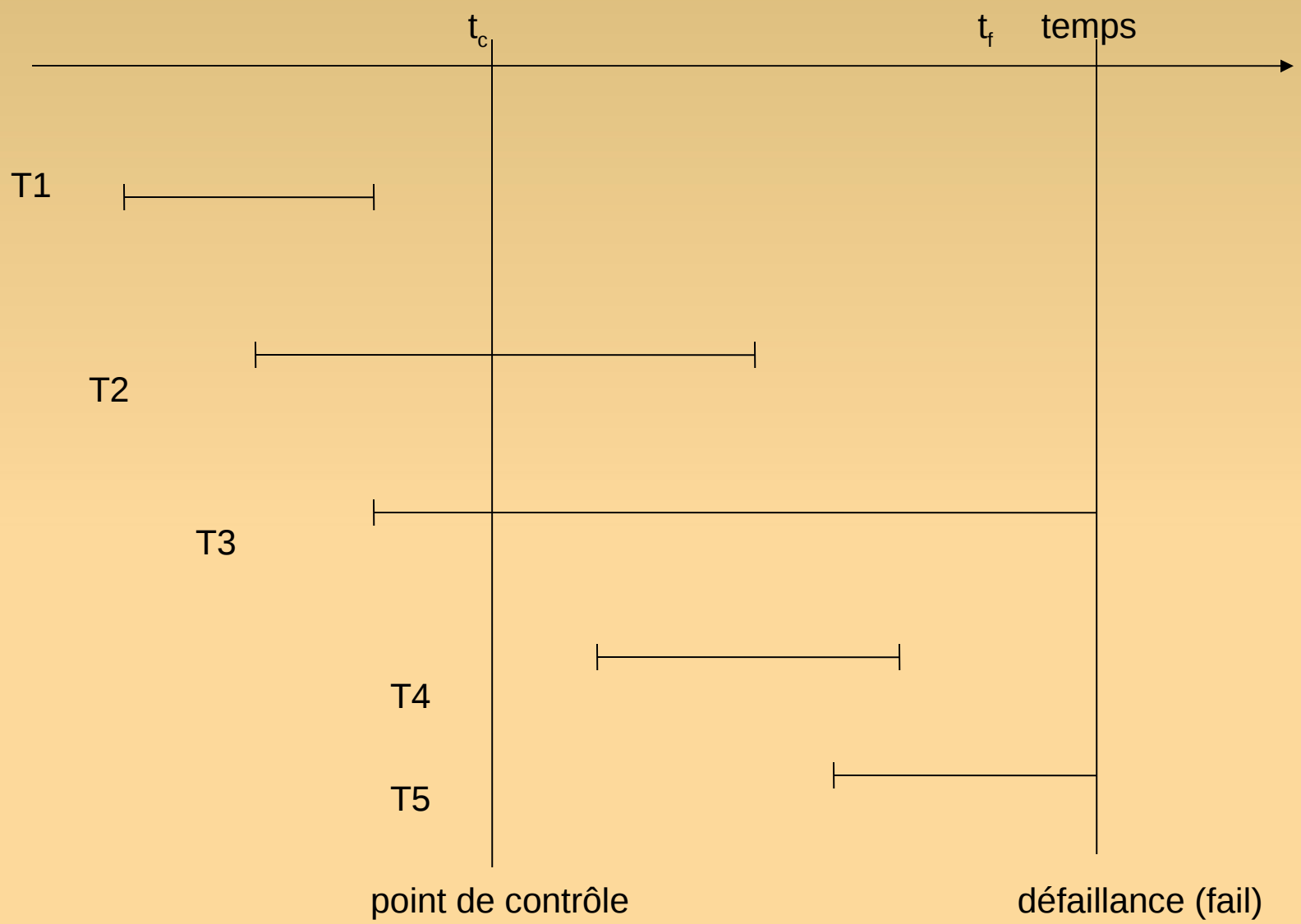
Dans les défaillances système, le point crucial est le contenu, **volatile, de la MC.** Les contenus des tampons de la BD en MC sont perdus. L ’état précis des transactions en cours d ’exécution au moment de la défaillance est inconnu. De telles transactions ne peuvent jamais se terminer par un succès et doivent être **annulées** (rollback).

De plus, il pourrait être nécessaire de **« rejouer »** certaines transactions lors du redémarrage du système. Ce sont celles qui ont effectué leur COMMIT avant la défaillance et dont les mises à jour n ’ont pas eu le temps d ’être inscrites dans la base.

Pour effectuer ce travail, le système utilise le contenu du **journal (log)**. A certains intervalles prédéfinis, le système positionne un **point de contrôle (checkpoint)** qui consiste :

- à écrire physiquement (écriture forcée) le contenu des tampons en MC sur les fichiers disque.
- À écrire physiquement le compte-rendu du point de contrôle dans le journal physique sur disque (log). Ce compte-rendu donne la liste de toutes les transactions qui étaient en cours d ’exécution lors du positionnement du point de contrôle.

Exemple :



A l ’instant t_c, il y a pose d ’un point de contrôle.

A l ’instant t_f, une défaillance survient.

Les transactions T1, …, T5 sont en cours d ’exécution.

- Les transactions de type T1 sont terminées avec succès (commit) avant t_c.
- Les transactions de type T2 ont débuté avant t_c, et se sont terminées avec succès (commit) après t_c et avant t_f.
- Les transactions de type T3 ont également débuté avant t_c, mais ne se sont pas terminées à la date t_f.
- Les transactions de type T4 ont débuté après t_c, et se sont terminées avec succès (commit) avant t_f.
- Les transactions de type T5 ont aussi débuté après t_c, mais ne se sont pas terminées à la date t_f.

Il devrait être clair que, lorsque le système est redémarré, les transactions de types T3 et T5 doivent être annulées, celles de types T2 et T4 doivent être rejouées. Les transactions de type T1 n 'interviennent pas car leurs màj ont déjà été transférées sur disque lors du chekpoint précédent (écriture forcée à la date t_c).

Par conséquent, lors du redémarrage du système, la procédure suivante est exécutée pour identifier les types de transactions T2 à T5:

1- Commencer par tenir 2 listes UNDO et REDO.

Initialiser la liste UNDO à toutes les transactions enregistrées dans le compte-rendu du point de contrôle le plus récent avant la panne. Initialiser la liste REDO à vide.

2- Faire une recherche en avant dans le journal en partant du point de contrôle.

3- Si un COMMIT T est rencontré, alors transférer la transaction T de la liste UNDO vers REDO.

4- A la fin du journal, la liste UNDO contiendra les transactions de types T3 et T5. La liste REDO contiendra les transactions de types T2 et T4.

Puis le système effectue un parcours en arrière du journal, annulant les transactions de la liste UNDO. Ensuite, il effectue à nouveau un parcours en avant, rejouant les transactions de la liste REDO.

Remarque : Remettre la BD dans un état cohérent en annulant le travail est appelé parfois *reprise en arrière*. De manière similaire, remettre la BD dans un état cohérent en rejouant des transactions est appelé parfois *reprise en avant*.

Enfin, lorsque l ’activité de reprise est terminée, le système devient prêt à accepter de nouveau travaux sur la BD.

Gestion des transactions

Accès concurrents

Gestion des transactions : Accès concurrents

- un grand nombre d'utilisateurs vont se connecter pour réaliser des opérations (lect, écr., màj, supp) simultanément

  => risque d ’incohérence des données + chute des perf.

- Trans = ens. d’opérations. Au minimum, une tr = 1 requête SQL.
- Début d ’une transaction : Begin,
- Fin d’une transaction : Commit (réussite) ou Rollback (échec).
- Une trans. a les propriétés ACID :
  - Atomicité : *tout ou rien*
  - Cohérence : *BD cohérente* -- transaction --> *BD cohérente*
  - Isolation : *les modif faites par une transac ne deviennent « visibles » par les autres qu ’à la suite de son Commit.*
  - Durabilité : *une fois qu ’une trans a Commité, ses modif deviennent permanentes dans la base, même si une panne survient.*

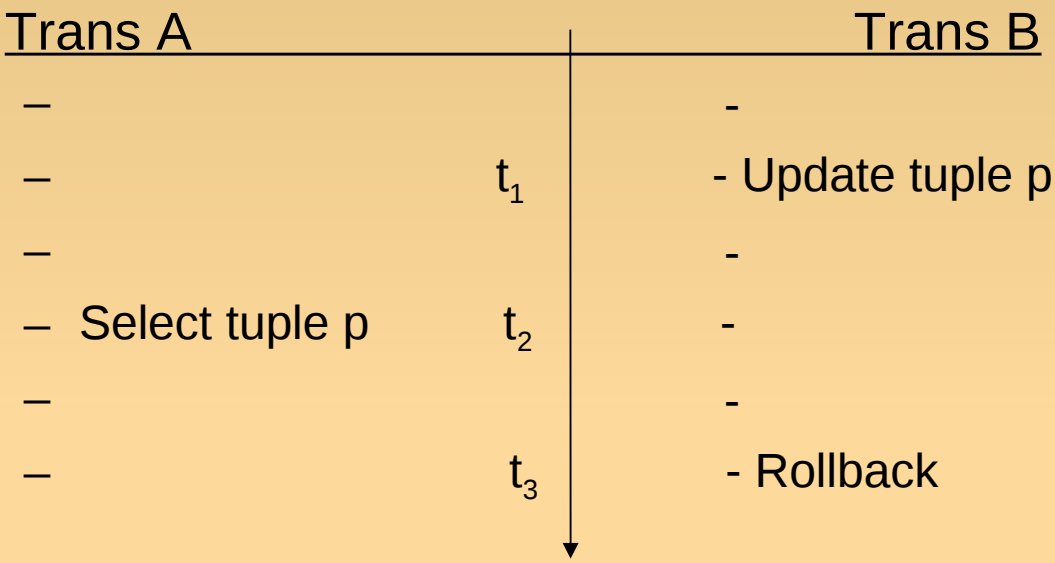
3 principaux problèmes lors de l ’exécution concurrente de transactions

- Perte de mise à jour : une transaction A récupère un tuple p à t_1; ensuite, la transaction B récupère le même tuple p à t_2; puis la transaction A met à jour le tuple p à t_3 (sur la base des valeurs de p lues à t_1); enfin, la transaction B met à jour le tuple p à t_4 (sur la base des valeurs de p lues à t_2, c-à-d identiques à celles lues à t_1).
- => la màj effectuée par A est perdue (écrasée par celle de B)

Trans A		Trans B
–		-
– Select tuple p	t_1	-
–	t_2	- Select tuple p
– Update tuple p	t_3	-
–	t_4	- Update tuple p

La mise à jour effectuée par A à l ’instant t_3 est perdue à l ’instant t_4.

Problème des dépendances non validées (1)

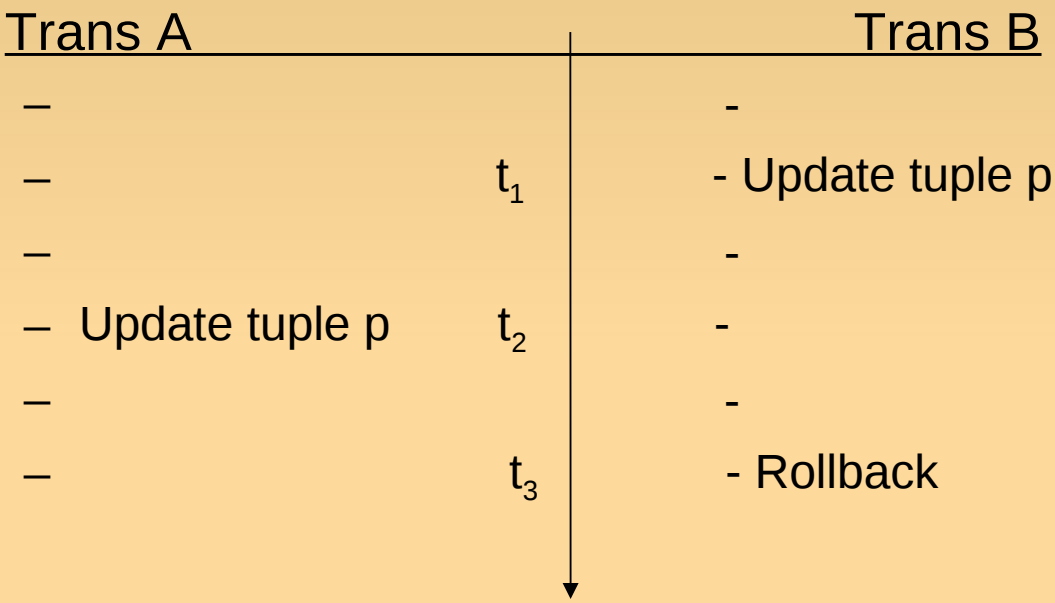


La transaction A devient dépendante d ’une modification non validée de la transaction B, i.e., la transaction A a lu des valeurs de p à t_2 qui ne sont plus valides à l ’instant t3 (car B a fait un Rollback)

(lecture salissante - dirty read)

Problème des dépendances non validées (2)

Cas encore pire (car A fait une màj du tuple p à t2, qui n 'est plus valide à t_3)



La transaction A devient dépendante d 'une modification non validée de la transaction B, i.e., la transaction A a mis à jour des valeurs de p à t_2 qui ne sont plus valides à l 'instant t3 (car B a fait un Rollback)

Problème de l ’analyse incohérente

Soient 3 comptes bancaires ACC1, ACC2 et ACC3, initialisés à

ACC1 <- 40, ACC2 <- 50 et ACC3 <- 30

Nous avons 2 transactions A et B; A fait le cumul des comptes dans la variable sum, et B transfère la somme de 20 du compte 3 vers le compte 1.

Voici une exécution possible de ces transactions en concurrence :

ACC1:40 ACC2:50 ACC3:30

Trans A		Trans B
Select ACC1: sum = 40	t_1	
Select ACC2: sum:=40+50	t_2	
	t_3	Select ACC3
	t_4	Update ACC3: ACC3 := ACC3-10
	t_5	Select ACC1
	t_6	Update ACC1: ACC1:=ACC1+10
	t_7	COMMIT
Select ACC3: sum=40+50+20	t_8	
sum=110 au lieu de 120 !!		

ACC1:50 ACC2:50 ACC3:20

Verrouillage

- Les pbs vus précédemment peuvent être résolue grâce à une technique de CC simple : le verrouillage.

- Idée de base : quand une transaction veut s ’assurer qu ’un objet (notamment un tuple) dont elle a besoin ne sera pas modifié de façon imprévisible, alors elle pose un verrou sur l ’objet. Le verrou a pour effet de verrouiller l ’accès à l ’objet par d ’autres transactions.

  Il les empêche en particulier de modifier l ’objet. La première transaction est ainsi sûre que l ’objet est stable aussi longtemps qu ’elle le souhaite.

Fonctionnement détaillé du verrouillage

- **1.** On suppose d ’abord que le système gère 2 types de verrous. Les verrous exclusif (X locks), et les verrous partagés (S, Shared locks), parfois appelés respectivement verrous d ’écriture et verrous de lecture. On suppose que les seuls verrous disponibles sont X et S, et que le seul objet ‘ verrouillable ’ est le tuple.

- **2.** Si la transaction A détient un verrou exclusif X sur le tuple p, alors la demande d ’un verrou de n ’importe quel type (X ou S) sur p faite par une autre transaction sera refusée

- **3.** Si la transaction A détient un verrou partagé S sur le tuple p, alors :
  - une demande d ’un verrou X sur p faitre par une autre transaction sera refusée
  - une demande d ’un verrou S sur p faite par une autre transaction B pourra être accordée (et la transaction B aura également posé un verrou S sur le tuple p)

Matrice de compatibilité des verrous X et S

A détient le verrou :

B demande le verrou	X	S	-
X	NON	NON	OUI
S	NON	OUI	OUI

Protocole d ’accès aux données (avec les verrous X et S)

- **1.** Une transaction qui souhaite extraire un tuple doit d ’abord obtenir un verrou S sur ce tuple

- **2.** Une transaction qui souhaite màj un tuple doit d ’abord obtenir un verrou X sur ce tuple. Si elle détient déjà un verrou S sur le tuple et qu ’elle désire le mettre à jour (séquence select/update), elle doit alors *promouvoir* le verrou S en X.

  Rmq : les demandes de verrous sont souvent implicites : Select (verrou S), update, delete (verrou X)

- **3.** Si une demande de verrou émise par B est refusée (car conflit avec un verrou détenu par A), alors B est mise en attente (jusqu ’à libération du verrou par A). Le système doit garantir que B n ’attende pas indéfiniment (verrouillage à vie).

- **4.** Les verrous exclisifs, X, sont conservésjusqu ’au COMMIT de la transaction (ou son ROLLBACK). En général, les verrous S également, mais pas toujours.

Eviter ou Résoudre le verrouillage à vie

- Souhaitable : détection puis suppression du blocage.

Détecter : voir si le **graphe d ’attente** des transactions (qui attend qui.) comporte un cycle.

Supprimer le blocage : choisir une transaction *victime* pour l ’annuler, et ensuite la redémarrer. On espère que l ’indéterminisme du système fera que la situation de blocage ne se renouvelle pas.

Notion de *time-out : certains système annulent et relancent une transaction qui reste inactive pendant un certain temps : le time-out*

Eviter les interblocages *(verrous mortels) :*

- obliger une transaction à demander tous ses verrous en même temps. Le système accepte de les donner tous ou aucun. Des données peuvent donc être verrouillées pour longtemps.

- fixer une relation d ’ordre sur les données et obliger les transactions à demander les verrous (sur les données) dans cet ordre.

Supprimer un interblocage :

Quand un interblocage survient, alors annuler une transaction pour la redémarrer plus tard. Pour cela, on affecte à chaque transaction une estampille unique, et on utilise ces estampilles pour décider en cas de conflit : si une transaction doit être mise en attente ou rejetée. Si elle est rejetée, elle garde la même estampille; ce qui la fait « vieillir ». Deux techniques existent pour régler les conflits entre transactions « jeunes » et « vieilles » :

1. **Wait-Die (attendre ou mourir)** : si une transaction T1 détient une donnée (verrou sur une donnée), que T2 réclame dans un mode non compatible, alors on autorise T2 à attendre si et seulement si T2 est plus vieille que T1. Si T2 est plus jeune, alors T2 est annulée et sera relancée avec la même estampille. Donc, plus T2 vieillit, plus elle est autorisé à attendre.

2. **Wount-Wait (Besser ou attendre)** : si une transaction T1 détient une donnée, que T2 réclame dans un mode incompatible, alors on autorise T2 à attendre ssi elle est plus jeune que T1. Si T2 est plus vieille, alors T1 est annulée (blessée), et elle est relancée avec la même estampille. Donc, plus T2 est vielle, plus elle est autorisée à accéder aux données.

- Rmq : avec les 2 techniques, la plus vielle des transactions ne peut être rejetée. Les 2 schémas évitent les situations d ’interblocage (deadlock).
- Exemple : 3 transactions T1 (5), T2 (10), T3 (15)

T1 plus jeune que T2 qui est plus jeune que T3.

- avec Wait-Die :

Si T2 réclame une donnée détenue par T1, alors T2 est autorisée à attendre (car plus vieille).

Si T2 réclame une donnée détenue par T3, alors T2 est rejetée (car plus jeune).

- avec Wount-Wait :

Si T1 réclame une donnée détenue par T2, alors T1 autorisée à attendre (car plus jeune que T2)

Si T3 réclame une donnée détenue par T2, alors T2 est avortée (annulée) car plus jeune que T3.

Résolution des 3 problèmes de concurrence

- **Pb de perte de màj :**

Trans A		Trans B
–		-
– Select tuple p	t_1	-
(obtention d'1 verrou S sur p)		
–	t_2	- Select tuple p
		(obtention d'1 verrou S sur p)
Update tuple p	t_3	-
(demande d'1 verrou X sur p)		
attente (car p est verrouillé en S par B)		
attente	t_4	- Update tuple p
attente		(demande d'1 verrou X sur p)
attente		*attente (car p est verrouillé en S par A)*
attente		*attente*

Aucune maj perdue mais un interblocage survient !

Pb des dépendances non validées (1)

Trans A		Trans B
–		-
–	t_1	- Update tuple p
		(obtention d ’une verrou X sur p)
–		-
– Select tuple p	t_2	-
(demande d ’un verrou S sur p)		
attente car p est verrouillé en X par B)		
–		-
–	t_3	- Rollback
		(libération du verrou X sur p)
– resume : Select p	t_4	
(obtention d ’un verrou S sur p)		
–		

La transaction A ne voit pas, à l ’instant t_2 ,la modif non validée de B (car elle doit se mettre en attente)

Pb des dépendances non validées (2)

Trans A		Trans B
–		-
–	t_1	- Update tuple p (obtention d ’une verrou X sur p)
–		-
– Update tuple p (demande d ’un verrou X sur p) attente car p est verrouillé en X par B)	t_2	-
–		-
–	t_3	- Rollback (libération du verrou X sur p)
– resume : Update p (obtention d ’un verrou X sur p)	t_4	
–		

La transaction A ne met pas à jour p (modif non validée) à l ’instant t_2 (car elle se met en attente)

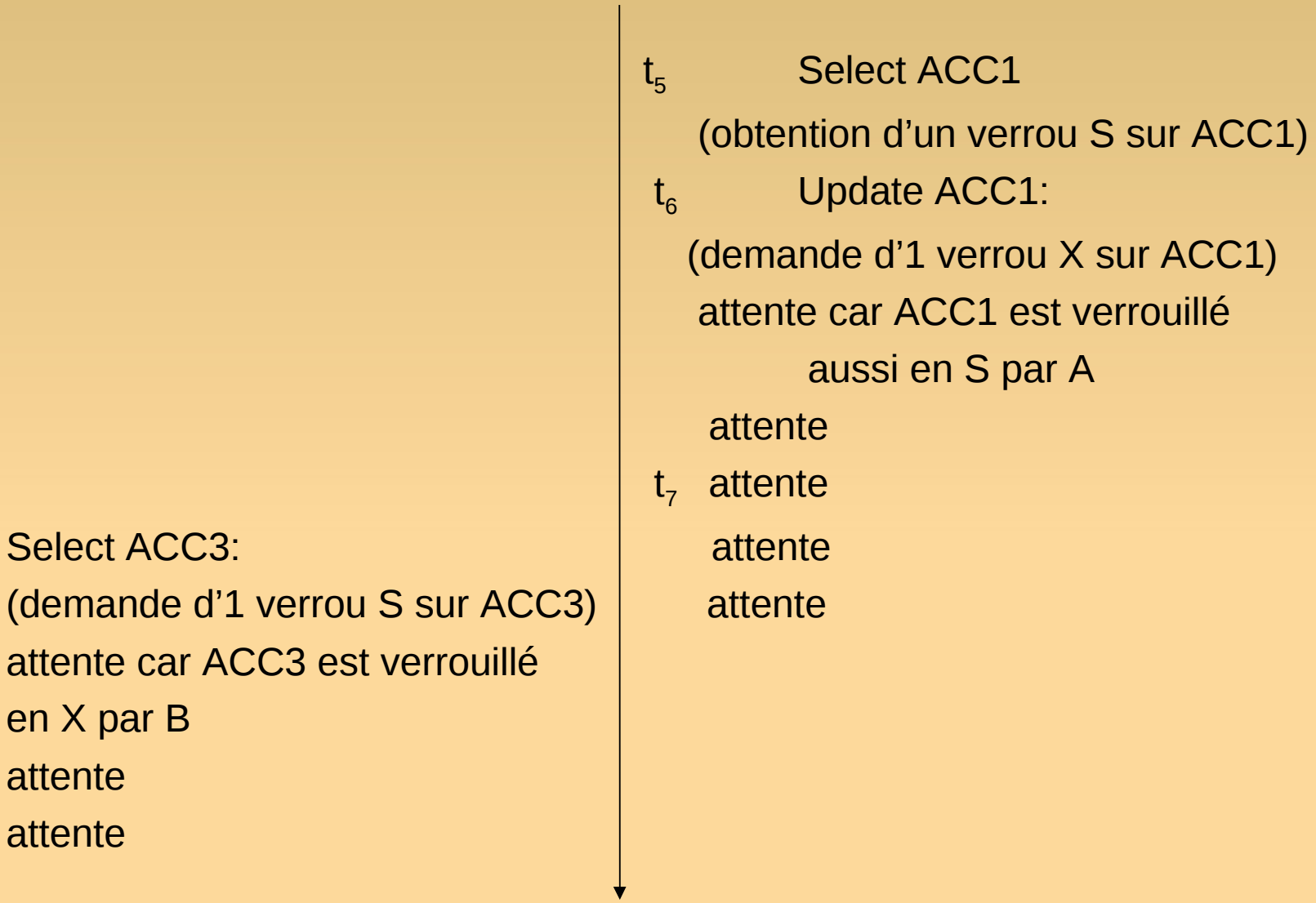
Rmq : A devient indépendant d ’une màj non validée (dans les 2 cas).

Problème de l'analyse incohérente (1)

ACC1:40 ACC2:50 ACC3:30

Trans A		Trans B
Select ACC1: sum = 40 (obtention d'1 verrou S sur ACC1)	t_1	
Select ACC2: sum:=40+50 (obtention d'1 verrou S sur ACC2)	t_2	
	t_3	Select ACC3 (obtention d'1 verrou S sur ACC3)
	t_4	Update ACC3: (obtention d'1 verrou X sur ACC3) ACC3 := ACC3-10

Problème de l ’analyse incohérente (2)



Rmq : L ’analyse incohérente est évitée, mais un blocage survient !

Notion de sérialisabilité

Il s ’agit du **critère de correction** généralement accepté pour le contrôle de concurrence, i.e., **une exécution entrelacée de transactions est considérée comme correcte si elle est *sérialisable***.

Une exécution (entrelacée) de transactions est sérialisable si elle produit le même résultat que leur exécution en série (l ’une après l ’autre).

=> justification :

* prise séparément, chaque transaction est supposée correcte (transforme un état cohérent de la BD en un autre état cohérent)

* l ’exécution des ces transactions les unes après les autres (séquentiel) est donc aussi correcte, puisque les transactions sont supposées indépendantes les une des autres

* Donc une exécution entrelacée est correcte, si elle est équivalente à une exécution en série (si elle est sérialisable).

Pour les exemples précédents, le problème est que, dans chaque cas, l ’exécution n ’était pas sérialisable. L ’effet du verrouillage était précisément de *forcer* la sérialisabilité.

Terminologie

Soit un ensemble de transactions.

Un ordonnancement = toute exécution (entrelacée ou non) de ces transactions.

Ordonnancement séquentiel : exécution des transactions les une après les autres (sans entrelacements).

A = {a1, a2, …, an}, B = {b1, b2, …, Bk}

Ordonnancement séquentiel : a1, …, an, b1, …, bk **ou** b1, …, bk, a1, …, an

Un exemple d ’ordonnancement entrelacé : b1, b2, a1, b3, a2, b4, …., bk, an)

Deux ordonnancements sont équivalents : s ’ils produisent les mêmes résultats quelque soit l ’état initial de la base.

Le concept de sérialisabilité fut introduit par Eswaran et al. 1976.

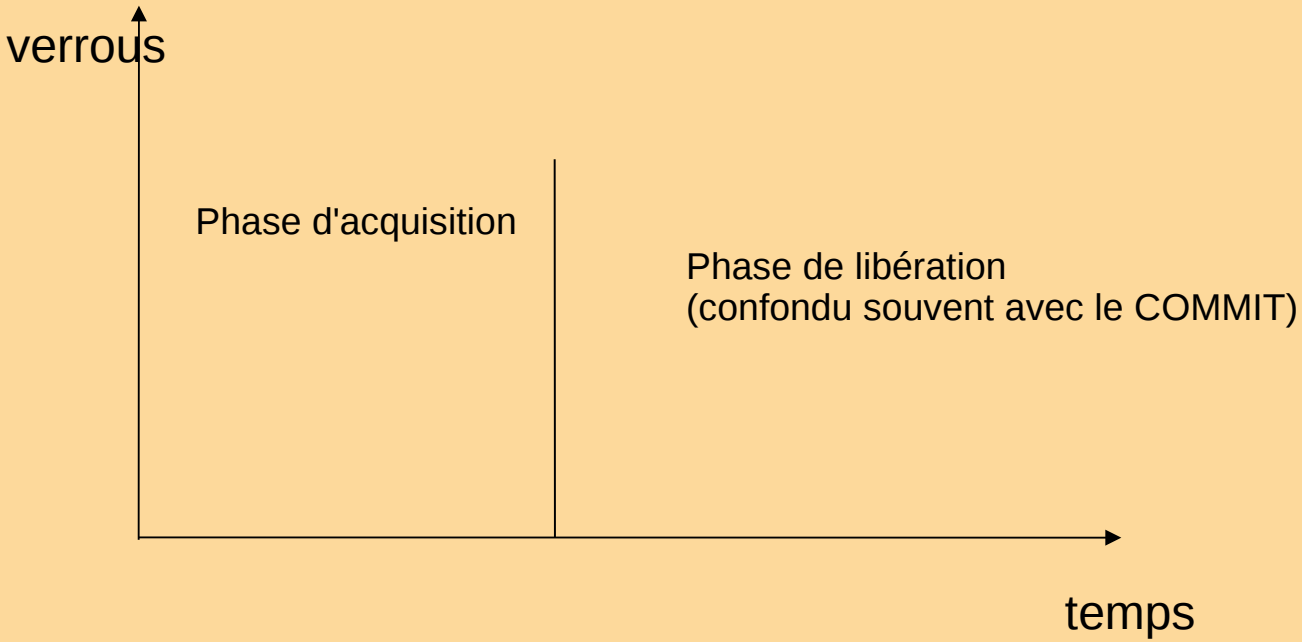
Le même article fournit également l ’énoncé d ’un théorème important : le théorème du verrouillage à 2 phases (2PL - Two-Phase Locoking) :

Thèorème : **si toutes les transactions satisfont au « protocole de verrouillage à 2 phases » alors tous les ordonnancements entrelacés sont sérialisables.**

Protocole de verrouillage à 2 phases :

1. Avant d ’agir sur un objet, (un tuple de BD, par exemple), une transaction doit obtenir un verrou sur cet objet.

2. Après l ’abandon d ’un verrou, une transaction ne doit plus jamais pouvoir obtenir de verrous.



Rmq : une transaction qui obéit à ce protocole a ainsi 2 phases : une phase d ’acquisition de verrous et une phase de libération. Dans la pratique, la phase de libération est souvent condensée en une seul opération : COMMIT (ou ROLLACK), à la fin de la transaction.

On peut considérer que le protocole présenté précédemment est un protocole de verrouillage à 2 phases.

Conclusion : Si une transaction A n ’est pas à 2 phases (n ’obéit pas au protocole 2PL), alors il est toujours possible de construire une transaction B qui peut s ’entrelacer avec A tel que l ’ordonnancement global n ’est ni sérialisable, ni correct.

En pratique, pour réduire les conflits sur les ressources (et améliorer les performances du SGBD), des systèmes autorisent des transactions qui ne sont pas à 2 phases (qui abandonnent prématurément les verrous - avant le commit), et obtiennent ainsi de nouveaux verrous. Mais cela présente des risques : en acceptant une transaction qui n ’est pas à 2 phases, on fait l ’hypothèse que pas une autre transaction ne coexistera avec A (sinon résultats erronés).

- **Le théorème est donc une condition *suffisante.***

- **Rmq** : la gestion de la concurrence peut être assurée sans réaliser de verrouillage mais par une technique de **gestion de versions**. Chaque requête est exécutée dans une copie virtuelle de la base qui l'isole des autres transactions. Mais d ’autres types de problèmes se posent pour gérer les différentes versions.

Résumé : les verrous

- 1. Verrouiller l'enregistrement modifié par la transaction 2.
- 2. Verrouiller les enregistrements lus par la transaction 1.
- 3. Verrouiller la table lue par la transaction 1.
- **Mais les verrous peuvent provoquer des situations de blocage :.**

Transaction 1	Transaction 2
Début	
Verrou sur A	Début
mise à jour dans A	Verrou sur B
lire B	mise à jour B
Attente (car B vérrouillé)	lire A (car A vérrouillé)
Attente	Attente

- Ce genre de blocage fatal est problématique et va être résolu par le sgbd.

Niveaux d ’isolation (1)

Est utilisé pour désigner pour décrire le ***degré d ’interférence*** qu ’une transaction donnée est prête à tolérer avec les autres transactions concurrentes.

Pour garantir la sérialisabilité, le seul degré d ’interférence toléré est le degré zéro (pas d ’interférences) !

Mais les systèmes réel autorisent des exécutions de transactions à des niveaux d ’isolation inférieurs à la sérialisabilité.

SQL comporte une instruction : SET TRANSACTION pour définir le niveau d ’isolation désiré pour les transaction (par défaut sérialisable, en général)

Niveaux d ’isolation :

READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALISABLE.

Le niveau le plus stricle est SERIALISABLE,

le moins strict est READ UNCOMMITTED.

Niveaux d ’isolation (2)

Si les transactions s ’exécutent au niveau SERIALISABLE, alors n ’importe quelle exécution entrelacée de ces transaction est garantie être sérialisable.

Si elles s ’exécutent avec un niveau d ’isolation inférieur, alors un de ces problèmes peut se poser :

Lecture salissante (dirty read) : possible avec READ UNCOMMITTED

T1 effectue une mise à jour sur un tuple p

T2 récupère ensuite ce tuple

T1 se termine par un ROLLBACK

=> la transaction T2 a alors observé un tuple qui n ’existe plus ou qui n ’a jamais existé (car T1 n ’a en fait jamais été exécutée)

Niveaux d ’isolation (3)

Lecture non renouvelable : possible avec READ COMMITTED et avec READ COMMITTED

T1 récupère un tuple p

T2 effectue ensuite une mise à jour de ce tuple

T1 récupère à nouveau ce tuple

=> la transaction T1 a alors récupéré le même tuple 2 fois et a observé des valeurs différentes !

Fantômes : possible avec tous les niveaux d ’isolation sauf SERIALISABLE

T1 récupère un ensemble de tuples (qui satisfont une condition)

T2 insère ensuite un tuple qui satisfait la même condition

Si T1 récupère de nouveau l ’ensemble de tuples qui fatisfont la condition, elle va observer un nouveau tuple qui n ’était pas auparavant (un fantôme !)

Niveaux d ’isolation (4)

Les différents niveaux d ’isolation sont définis selon le type de violation de la sérialisabilité qu ’ils autorisent. La figure suivante en donne un récapitulatif.

PROBLEME *Niveau d ’isolation*	LECT. SALIS.	LECT. NON RENOUV.	FANTOME
Lect. non validée **(read uncommitted)**	**oui**	**oui**	**oui**
lecture validée **(read committed)**	**non**	**oui**	**oui**
lect. renouvelable	**non**	**non**	**oui**
sérialisable	**non**	**non**	**non**

Niveaux d'isolation : résumé

- Pour les niveaux les moins restrictifs, toutes les situations d'incohérence ci-dessus peuvent se produire. Dans les plus restrictifs, toutes ces incohérences sont évitées.

- Niveau **READ COMMITED** : chaque transaction garantit au moins que le cas des lectures salissantes ne peut se produire. Cas de tous les SGBD modernes. Ce niveau n'est pas très sûr pour assurer la cohérence.

- Niveau **SERIALIZABLE** : niveau fourni par le système qui évite les 3 types d'incohérence et signifie que les requêtes sont exécutées comme si elles étaient lancées les unes après les autres.

- ***Rmq*** *: Un inconvénient avec ces transactions très sûres est qu'elles font baisser les performances du système. Des transactions commencées bloquant l'accès aux données devront être terminées avant que la transaction sérialisable ne se poursuive.*

- *Il est probable que le passage dans l'état sérialisable soit difficile à réaliser et que plusieurs essais soient nécessaires. Si une transaction A tente de faire une modification ou suppression alors qu'une autre instruction B réalise aussi une suppression ou modification sur le même enregistrement, A s'annule avec un message d'erreur.*

- Avec SQL : la modification du type de transaction est réalisée avec :

- SET TRANSACTION
  [ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE }]
  [READ WRITE | READ ONLY]

Verrous explicites

- Dans de nombreux SGBD, la résolution de conflits est évitée grâce à un système de verrouillage des tables ou enregistrements. Les verrous sont posés pour une transaction et persistent jusqu'à sa fin. La commande est :
  - LOCK [TABLE] name [, ...] [IN lockmode MODE]

  où lockmode peut être :

  ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE

- **Rmq** : Les verrous sont posés au niveau de la table avec cette commande. Le niveau le plus élevé est ACCESS EXCLUSIVE qui assure que l'accès est réservé uniquement à la transaction en cours. C'est le niveau demandé pour la modification de la structure d'une table par exemple.
- Le verrouillage au niveau enregistrement est réalisé par la commande
  - SELECT FOR UPDATE

- Cette commande se met en attente de libération de verrou si les enregistrements sont déjà verrouillés par une autre transaction.

Verrous intentionnels

Granularité de verrouillage : lverrous peuvent porter sur des éléments plus ou moins gros qu ’un tuple (une relation, la BD, une valeur d ’attribut, …).

Il faut trouver un compromis entre une concurrence élevée (granularité fine) et une concurrence faible (grosse granularité -relation, par exemple).

Plus la granularité est grossière, moins on a besoin de positionner et tester des verrous (moins de charge). Par exemple, si verrou sur une relation entière, alors pas besoin de positionner des verrous sur chaque tuple.

==> Donc réduction du nombre de verrous, MAIS aucune autre transaction concurrente ne peut y accéder (en mode incompatible).

Lors de la demande d ’accès à 1 tuple, et pour ne pas avoir à tester tous les tuples d ’une relation pour savoir si ce tuple particulier est verrouillé, on introduit un protocole de **verrouillage intentionnel.**

Ce protocole impose qu ’aune transaction ne doit pouvoir accéder à un tuple d ’une relation sans avoir obtenu au préalable le verrou adéquat sur la relation qui contient ce tuple.

Verrous intentionnels 2)

La détection de conflit dans l ’exemple précédent consiste à voir si la transaction demandeuse (pour verrouiller un tuple) n ’a pas de verrou conflictuel au niveau de la relation.

Les verrous X et S ont un sens aussi bien pour la relation que pour les tuples.

3 nouveaux verrous intentionnels sont introduits qui ont un sens pour une relation, mais pas pour un tuple : verrou intentionnel partagé (IS), verrou intentionnel exclusif (IX), verrou intentionnel exclusif et partagé (SIX).

Verrous intentionnels 3)

En supposant qu ’une transaction T demande un verrou de type indiqué (IS, IX, S, SIX, X), sur la relation R, on a alors :

IS : T veut positionner des verrous S sur des tuples de R (pour garantir leur stabilité pendant l ’exécution de T).

IX : Identique à IS + le fait que T pourrait mettre à jour des tuples particuliers de R (en positionnant des verrous X sur ces tuples S : T peut tolérer des lectures concurrentes de R (verrous S sur des tuples de R) mais pas des mises à jour. T elle-même n ’effectue aucune mise à jour de R.

SIX : combine S et IX : T peut tolérer des lectures concurrentes de R (verrous S sur des tuples de R) mais pas des mises à jour + le fait que T elle-même peut effectuer des mises à jours sur des tuples particuliers de R. (en posant des verrous sur ces tuples).

X : T ne tolère aucun accès concurrent à R. T elle-même pourrait mettre à jour ou pas des tuples de R.

Matrice de compatibilié

A détient le verrou

B demande le verrou

	X	SIX	IX	S	IS	pas de verrou
X	N	N	N	N	N	O
SIX	N	N	N	N	O	O
IX	N	N	O	N	O	O
S	N	N	N	O	O	O
IS	N	O	O	O	O	O

Protocole de verrouillage intentionnel

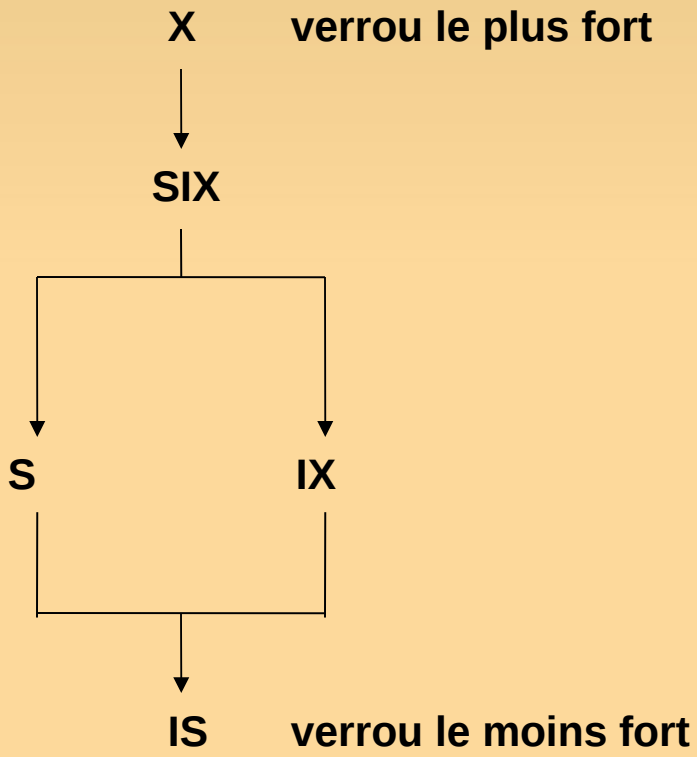
- **1.** Avant qu ’une transaction ne puisse obtenir un verrou S sur un tuple, elle doit d ’abord obtenir un verrou IS (ou plus fort) sur la relation contenant ce tuple.
- **2.** Avant qu ’une transaction ne puisse obtenir un verrou X sur un tuple, elle doit d ’abord obtenir un verrou IX (ou plus fort) sur la relation contenant ce tuple. La force relative d ’un verrou :

Rmq :

- si une demande de verrou échoue pour un type de verrou donné, elle échouera nécessairement pour un type de verrou plus fort.

- les forces relatives des verrous S et IX ne sont pas comparables.

- en pratique, les verrous intentionnels sur les relations sont obtenus implicitement



- Par exemple, pour une transaction en lecture seule, il est probable que le système obtienne un verrou IS sur chaque relation accédée par la transaction (pour que les tuples qu ’elle lit restent stables).
- Pour une transaction en mise à jour, il est probable qu ’elle obtienne un verrou IX.

- **Rmq** : il existe un compromis entre minimiser la charge du système en maximisant la concurrence : le verrouillage **en escalier**. L ’idée de base est que le système remplace plusieurs verrous au niveau des tuples par un verrou intentionnel dès qu ’il atteint un certain seuil (nombre de tuples verrouillés).