École Doctorale 2008/2009

Systèmes de Bases de Données avancés

École nationale Supérieure d'Informatique (ESI)

Objectifs

- Concepts de base
- Architecture
- Nouvelles générations

Plan

1. Notions de base
2. Traitement des requêtes
3. Contrôle de concurrence
4. Recouvrement
5. Parallélisme et distribution
6. Nouveaux systèmes post-relationnels
7. Quelques Projets

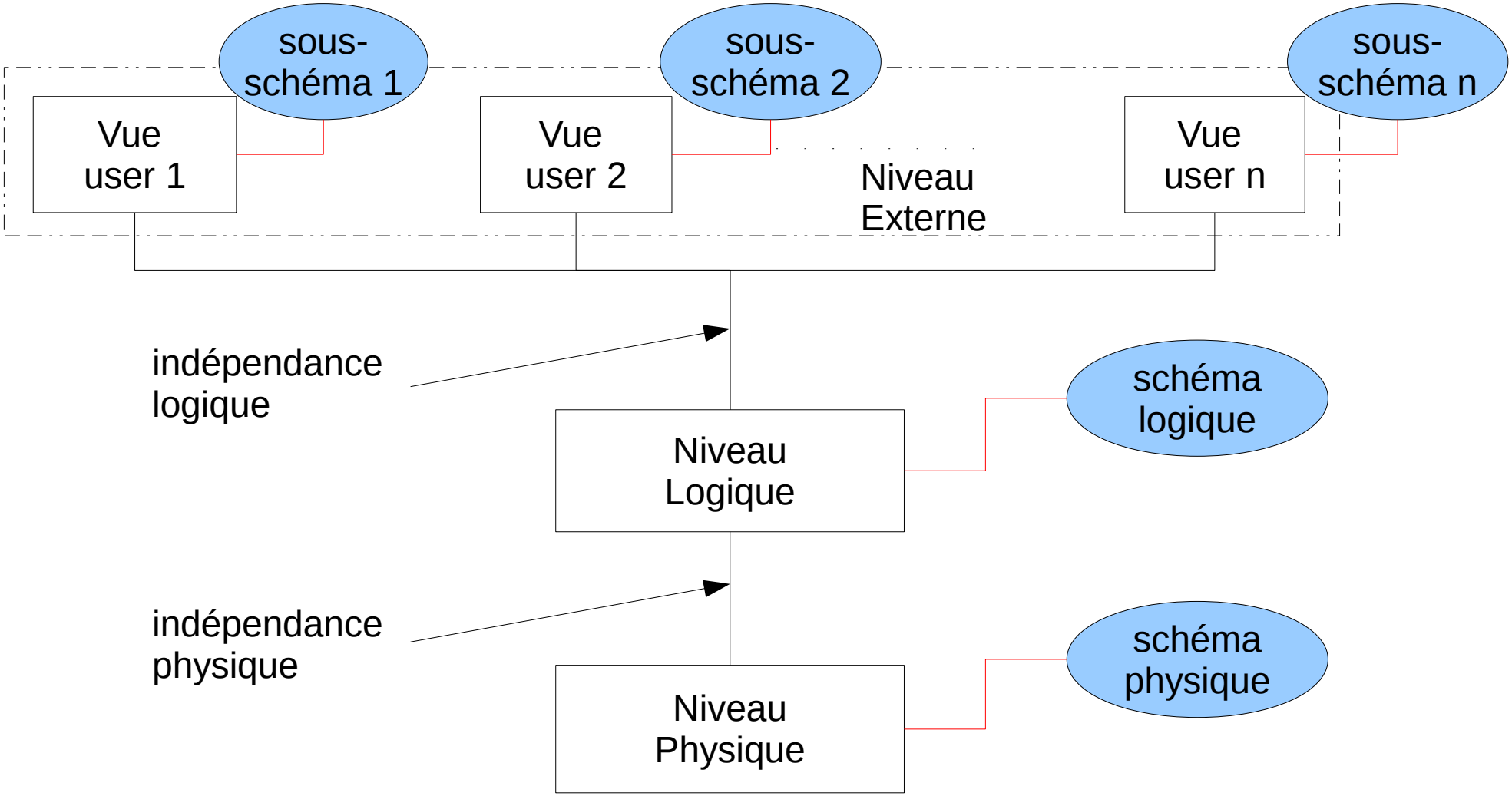
1.1 Définitions

- SGBD (Système de Gestion de Bases de Données)
  - « grand » volume de données liées entre elles (= **BD**)
  - ensemble de programmes qui gèrent ces données (= **Système de gestion**)
- Objectifs
  - efficacité (très important)
  - sécurité
  - facilité d'utilisation

1.2 Traitement de fichiers (Pb)

- Redondance et inconsistance des données
  - certaines info se trouvent sur plusieurs fichiers
- Difficulté d'accès aux informations non prévues
  - nécessité d'écrire de nouveaux prog. d'accès
- Dépendance : rep. interne / Applications
  - changement de structure -> re-programmation des App
- Atomicité et pb de concurrence
  - erreur, pannes, accès concurrents -> introduisent des inconsistances

1.3 Abstraction des données



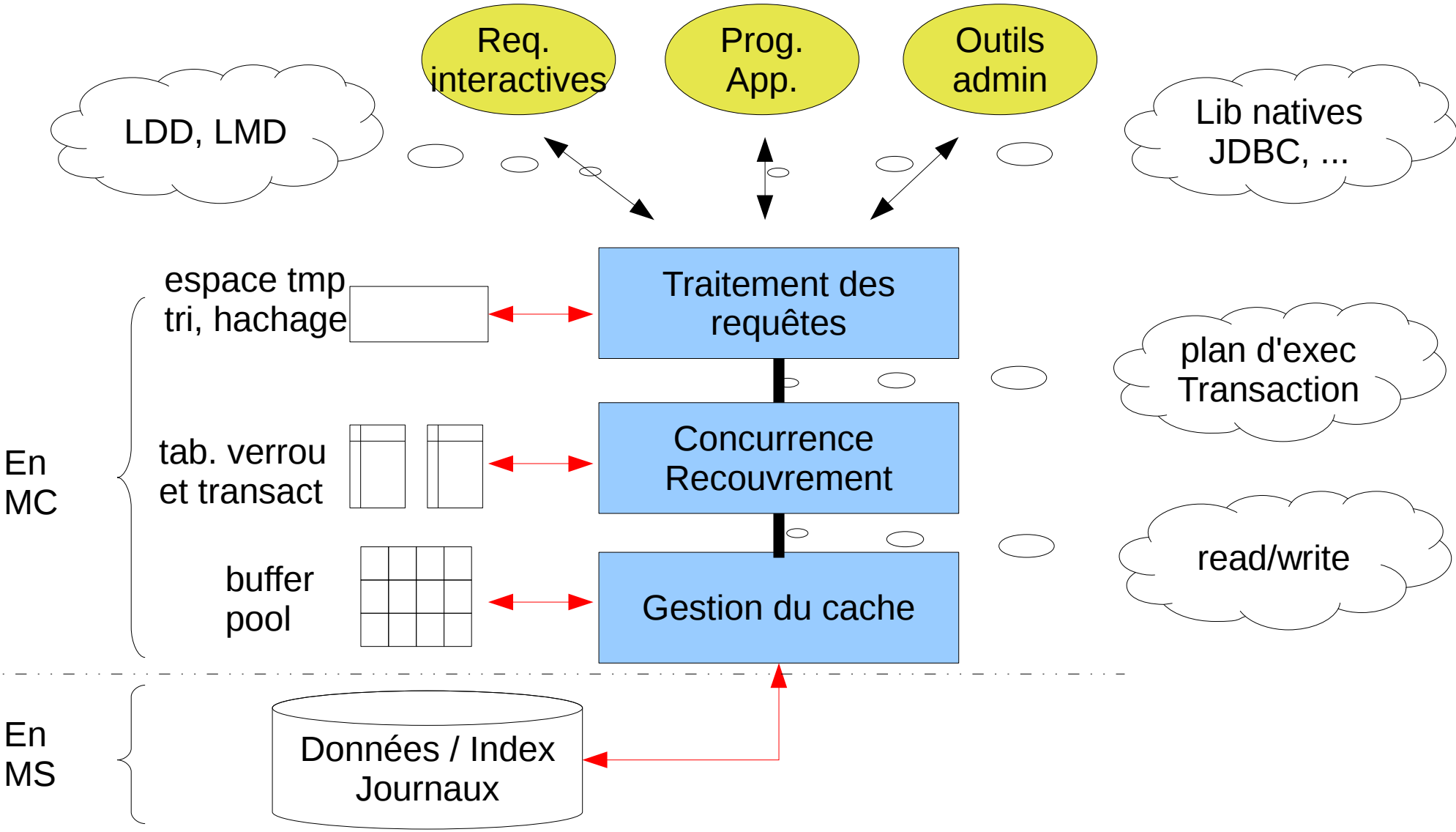
1.4 Modèles de données

- Modèle de données

  Outils conceptuels pour décrire les données, les liens, la sémantique et les contraintes.

- Exemples
  - Le modèle Entités-Associations
  - Le modèle Relationnel
  - Le modèle Objet
    - Le modèle Relationnel-Objet
  - Anciens modèles:
    - Hiérarchique, Réseau (CODASYL)

1.5 Architecture système



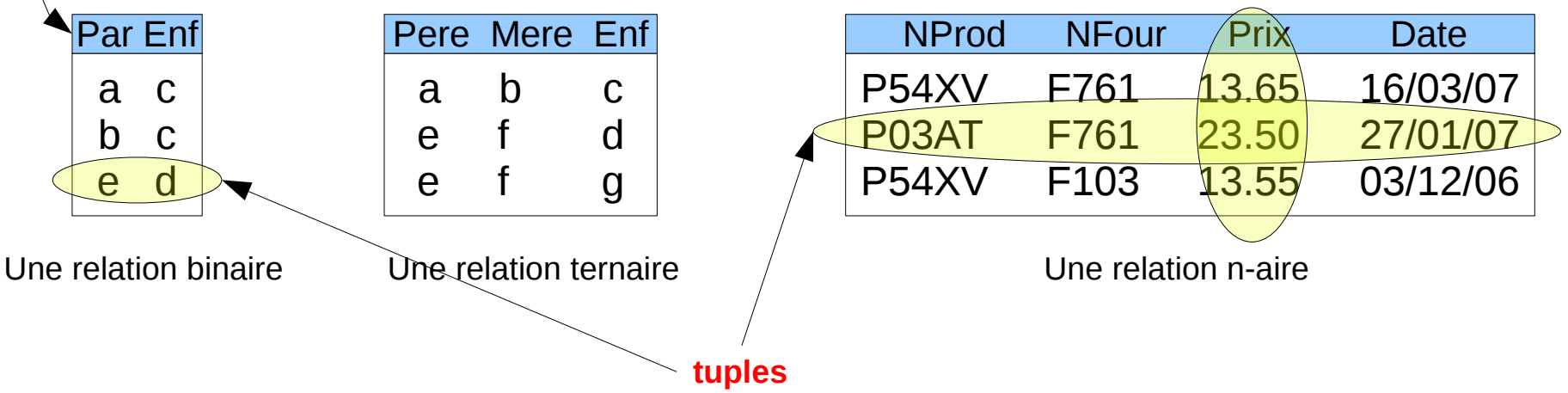
1.6 Le modèle relationnel

- Populaire
  - Utilisé par la majorité des systèmes actuels
  - Simple et bien formalisé

- Plan
  - Structure d'une BD relationnelle
  - Algèbre relationnelle et extensions
  - Calculs relationnels
  - SQL

1.6 Le modèle relationnel

- BD = ensemble de tables relationnelles
- Table T(A1:D1, A2:D2, ... An:Dn)
  - sous-ens du produit cartésien des domaines de ses constituants. $T \subseteq \{D1xD2x...Dn\}$
  - Une table est donc une Relation



1.6 Le modèle relationnel
- clés -

- Super-clé
  - Ens d'attributs identifiant chaque tuple
- Clé candidate
  - Super-clé minimale
- Clé primaire
  - La clé candidate choisie par le concepteur
- Exemple:

  Employe(<u>NSS</u>, Nom, Adr, Specialite, Salaire)

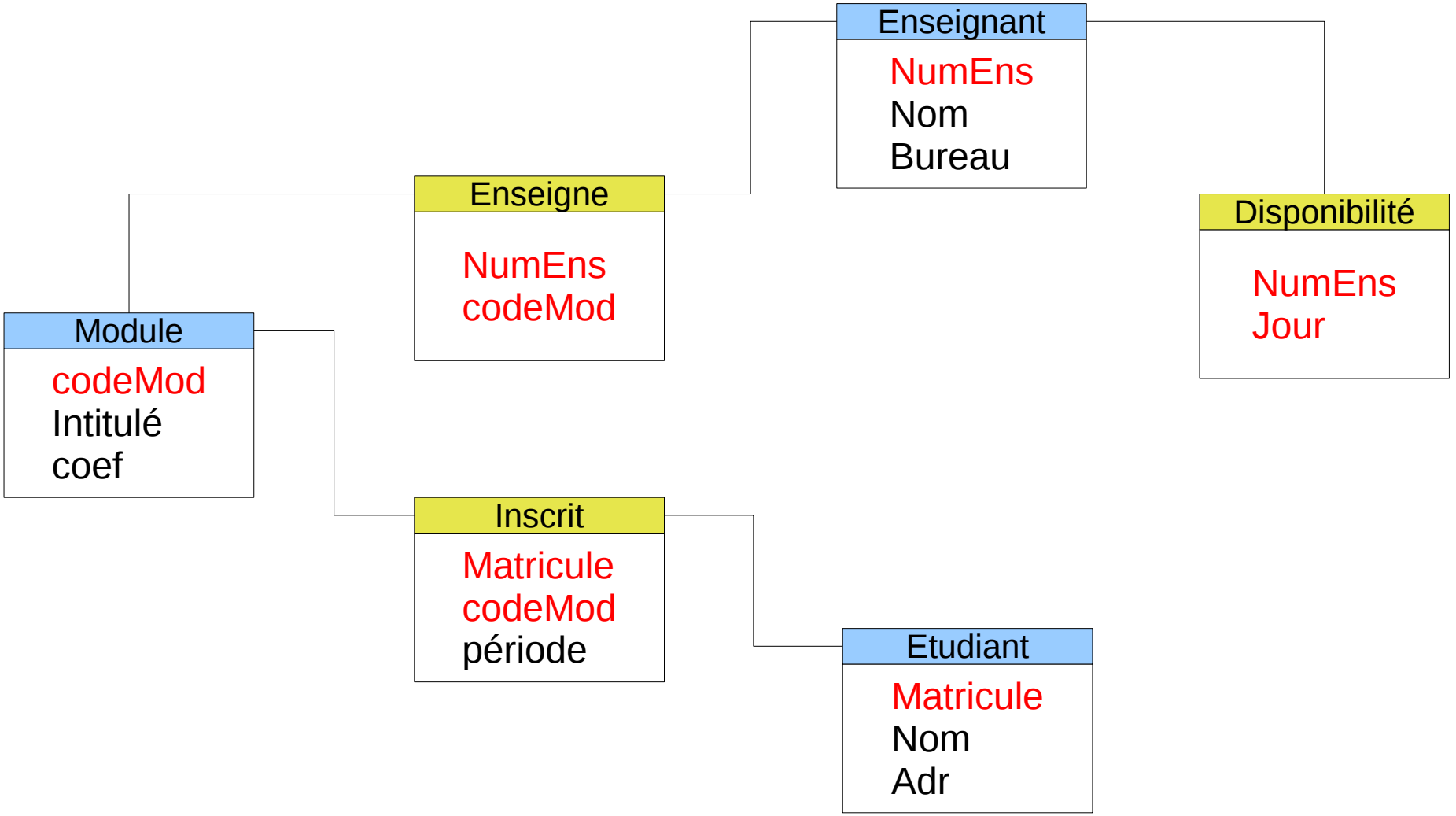
  {NSS, NOM} --> super-clé

  {NSS} --> clé candidate

1.6 Le modèle relationnel
- conception (informelle) de schémas -

- Il y a des tables qui représentent des entités avec leurs propriétés
  - Patient(NumP, Nom, Adr)
  - Pathologie(code, description)
- Il y a des tables qui représentent des associations entre entités
  - Malade(NumP, code, date)
- Il y a des tables qui représentent des champs multi-valués
  - Symptomes(code, symptome)

1.6 Le modèle relationnel
- exemple : diagramme de schéma -



1.6 Le modèle relationnel
- Normalisation -

- Dépendances fonctionnelles

  x --> y / x et y des ens. d'attributs, t1 et t2 des tuples
  (t1[x] = t2[x] implique t1[y] = t2[y])

- Forme Normale de Boyce-Codd (BCNF)

  R est en BCNF si pour toute dep. (non triviale) x-->y, x est une super-clé de R

- 3e Forme Normale

  même chose que BCNF ou alors tout attr dans (y-x) est contenu dans une clé candidate

1.6 Le modèle relationnel
- Algèbre -

- Langage de requêtes procédural
  - On doit spécifier comment trouver le résultat
- Chaque opération a 1 ou 2 relations en entrée et 1 relation en sortie
- Une requête est une expression (composition) d'opérations algébriques

1.6 Le modèle relationnel
- Algèbre : opérations de base -

opérateurs unaires

- Sélection ou Restriction

  σ (cond / R)
- Projection

  $\prod$ (liste_attr / R)
- Renommage

  r (R1 / R2)

opérateurs binaires

- Union

  R1 $\cup$ R2
- Différence

  R1 - R2
- Produit cartésien

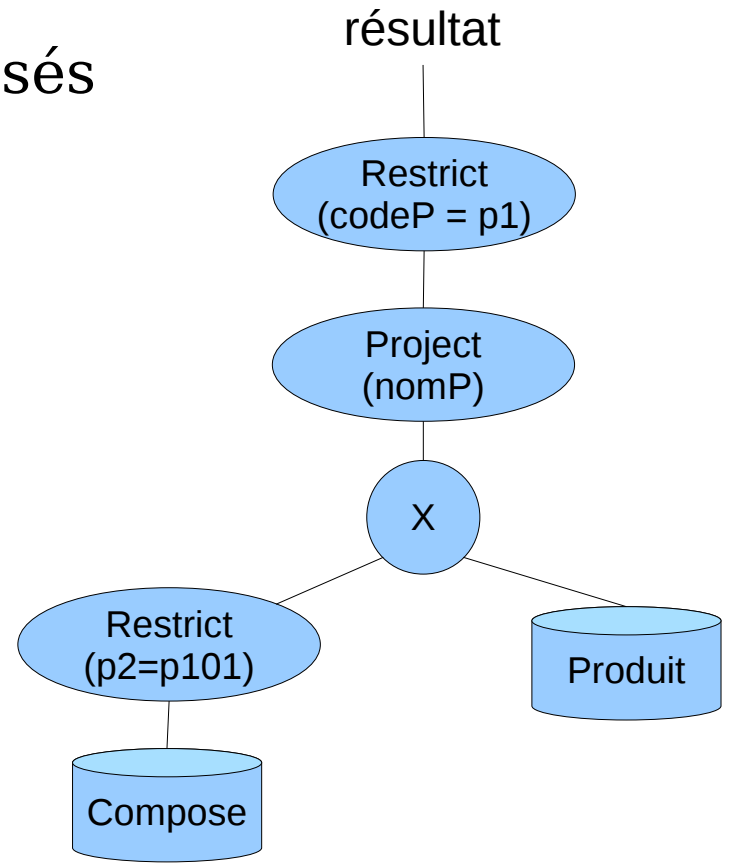
  R1 x R2

1.6 Le modèle relationnel
- Algèbre : exemple -

- Produit(codeP, NomP, description)
  Compose(p1, p2, qte)
  // le produit 'p1' est composé de 'qte' sous-prod 'p2'

  Donner le nom des produits composés par le sous-produit de code 'P101'

  R1 = σ (p2='P101' / Compose)
  R2 = Produit x R1)
  R3 = σ (codeP=p1 / R2)
  Résultat = ∏ (NomP / R3)



1.6 Le modèle relationnel
- Algèbre : autres opérations -

- Intersection (R et S de même schéma)

  $R \cap S = R - (R - S)$

- Jointures (naturelle – cnd implicite, théta – cnd explicite)

  $R \otimes_{(cnd)} S = \sigma (cnd / R \times S)$

- Division (schéma de S $\subset$ dans schéma de R)

  $R_{(a1,a2,..b1,b2,...)} \div S_{(b1,b2,...)} = T_{(a1,a2,..)}$

  $= \Pi(a1,a2,.../ R) - \Pi(a1,a2,.../ ((\Pi(a1,a2,.../ R) \times S) - R))$

  un tuple t est dans T ssi
  - t est dans Π(a1,a2,.../ R) et,
  - pour chaque tuple ts dans S, il existe un tuple tr dans R :
    tr[b1,b2,...] = ts et tr[a1,a2,...] = t

1.6 Le modèle relationnel
- Algèbre étendue -

- Projection généralisée
  - Π(exp1, exp2, .../ R) exp : attr ou val calculée
- Fonctions d'agrégation et groupage
  - Gr(a1,a2,.../ F1(b1), F2(b2), .../ R)
  - Ex : meteo(ville, temp, humid, precipitations, date)
    donner pour chaque ville les températures min et max
    Gr(ville / min(temp) as T0, max(temp) as T1 / meteo)

1.6 Le modèle relationnel
- Algèbre étendue : OUTER JOIN -

Rajoute au résultat de la jointure les tuples ne vérifiant pas la condition de jointure avec NULL dans les attr vides

LEFT OUTER JOIN : $R \text{-}\otimes_{(cnd)} S$

les tuples de R ne vérifiant pas cnd seront rajoutés au résultat avec des NULL à la place des attr de S

RIGHT OUTER JOIN : $R \otimes\text{-}_{(cnd)} S$

les tuples de S ne vérifiant pas cnd seront rajoutés au résultat avec des NULL à la place des attr de R

FULL OUTER JOIN : $R \text{-}\otimes\text{-}_{(cnd)} S$

les tuples de R et de S ne vérifiant pas cnd seront rajoutés au résultat avec des NULL à la place des attr manquant

1.6 Le modèle relationnel
- Algèbre étendue : NULL -

Les valeurs manquantes sont représentées par NULL

Toute opération (+, -, * ou /) avec un NULL rend NULL

Toute comparaison (<, <=, >, >=, =, ..) avec un NULL rend NULL

P1	P2	P1 and P2	P1 or P2
V	null	null	V
F	null	F	null
null	null	null	null

not null = null

1.6 Le modèle relationnel
- Algèbre étendue : Mises à jour -

- Insertion

  $R := R \cup E$

  rajoute à R le résultat de l'expression Alg E

- Suppression

  $R := R - E$

  enlève de R les tuples vérifiant E

- Modification

  $R := \Pi(exp1, exp2, .../ \sigma\ (cond / R)) \cup (R - \sigma\ (cond / R))$

  met à jour les tuples vérifiant la condition 'cond' avec les nouvelles valeur d'attribut : exp1, exp2 ...

1.6 Le modèle relationnel
- Calcul Relationnel à var. tuple -

Langage basé sur la logique du 1er ordre où les var portent sur des tuples

Forme générale : { t / P(t) }

l'ensemble des tuples t qui vérifient la formule P(t)

Formule :

- tout atome est une formule:
  $t \in R$, t[a] op t[b], t[a] op cste / op : <, >, =, ...
- si P est une formule, alors (P), $\neg P$, $(\exists t \in R\ P)$, $(\forall t \in R\ P)$ sont aussi des formules
- si P et Q sont des formules, alors $P \wedge Q$, $P \vee Q$, $P \Rightarrow Q$ sont aussi des formules

1.6 Le modèle relationnel
- Calcul à var. tuple : exemples -

Produit(codeP, NomP, prix) Compose(p1, p2, qte)

// 'p1' est composé de 'p2' avec une quantité 'qte'

- Trouver tous les code, nom et prix des produits coûtant moins que 100 DA

  $\{ t / t \in Produit \wedge t[prix]<100 \}$

- Trouver tous les noms des produits coûtant moins que 100 DA

  $\{ t / \exists s \in Produit (s[prix]<100 \wedge t[nom] = s[nom]) \}$

- Trouver les noms de produits ainsi que les quantités nécessaires rentrant dans la composition du produit de code 'P326'

  $\{ t / \exists c \in Compose$

  $(c[p1]='P326' \wedge t[qte] = c[qte] \wedge$

  $\exists p \in Produit (c[p2]=p[codeP] \wedge t[nomP] = p[NomP])) \}$

1.6 Le modèle relationnel
- Calcul à var. tuple : exemples -

Produit(codeP, NomP, prix) Compose(p1, p2, qte)

// 'p1' est composé de 'p2' avec une quantité 'qte'

- Trouver les codes de produits qui composent tous les produits ayant un prix > 1000 DA.

$$\{ t / \exists c \in Compose (c[p2]=t[p2] \wedge \forall p \in Produit (p[prix] > 1000 \Rightarrow c[p1] = p[codeP])) \}$$

- S'il n'y a pas de produit avec un prix > 1000, tous les codes (p2) seront dans la réponse
  on peut rajouter une condition qui force l'existence d'au moins un produit avec un prix > 1000

1.6 Le modèle relationnel
- Calcul à var. tuple : exp saines -

Certaines expressions produisent un résultat infini

$\{ t / \neg (t \in R) \}$

Une expression $\{ t / P(t) \}$ est « saine » ou « correcte » si toutes les valeurs qui apparaissent dans le résultat appartiennent au « domaine » de P(t).

dom(P(t)) : l'ensemble des valeurs qui apparaissent dans P(t) et dans les relations référencées dans P(t).

dom($\neg (t \in R)$) = l'ens des valeurs apparaissant dans R. Or il existe des tuples dans $\{ t / \neg (t \in R) \}$ formés par des valeurs n'appartenant pas à R.
Donc l'exp $\{ t / \neg (t \in R) \}$ n'est pas saine.

1.6 Le modèle relationnel
- Calcul Relationnel à var. Domaine -

Langage basé sur la logique du 1er ordre où les var portent sur les domaines d'attributs

Forme générale : { <x1,x2,...xn> / P(x1,x2,...xn) }

l'ensemble des tuples qui vérifient la formule P(x1,x2,...)

Formule :

- tout atome est une formule:
  <x1,x2...> $\in$ R, x op y, x op cste / op : <, >, =, ...
- si P est une formule, alors (P), $\neg$P, ($\exists$x P),($\forall$x P) sont aussi des formules
- si P et Q sont des formules, alors P$\wedge$Q, P$\vee$Q, P$\Rightarrow$Q sont aussi des formules

1.6 Le modèle relationnel
- Calcul à var. Domaine : Exemples -

Produit(codeP, NomP, prix)

Compose(p1, p2, qte)
// 'p1' est composé de 'p2' avec une quantité 'qte'

- Trouver les code, nom et prix des produits coûtant moins de 500 DA
  {<x,y,z> / <x,y,z> $\in$ Produit $\wedge$ z < 500 }
- Trouver les noms de produits coûtant 100 DA
  {<y> / $\exists$ x,z (<x,y,z> $\in$ Produit $\wedge$ z = 100) }

1.7 SQL

- Forme simple -

SELECT exp1, exp2, ...
   FROM R1, R2, ...
   WHERE <Cond>

= Π(exp1, exp2, ... / σ (Cond / R1 x R2 x ...))

Exemples:

SELECT NomP FROM Produit WHERE codeP='P653';

SELECT NomP, qte
   FROM Produit, Compose
   WHERE codeP=P2 AND P1='P326';

1.7 SQL

- Forme simple -

SELECT <Liste_select>

FROM <Exp_table> [specification_tri]

```
SELECT * FROM table1;
SELECT a, b + c FROM table1 ORDER BY a DESC;
SELECT 3 * 4;
SELECT random();
```

1.7 SQL

- Liste de selection -

<exp_val> , <exp_val> , ... / *

<exp_val> = Expression de valeurs :

- Constante / ref de colonne / param fonction
- <exp> opérateurs <exp> (a + 3)
- Appel de fonction
- Expression d'agrégation (min, max, count ...)
- sous requête modulable (retourne un scalaire)

1.7 SQL

- Liste de selection -

« Lister pour chaque pays le nombre de citadins »

pays(nom, superficie, population, …)

villes(nom, etat, pop, …)

```
SELECT nom, (SELECT sum(pop) FROM villes
                    WHERE villes.etat = pays.nom)
FROM pays;
```

1.7 SQL
- Expression de tables -

FROM <tab_ref>, ...

[WHERE | GROUP BY | HAVING]

Rôle: Produire une table intermédiaire sur laquelle porte la <Liste_select>

<tab_ref> = nom de table (base ou dérivée)

table dérivée = sous_req, tables jointes, ...

1.7 SQL
- tables jointes -

T1 CROSS JOIN T2

- équivalent à : T1 , T2
- réalise le produit cartésien entre T1 et T2
- La table résultat contiendra tous les attributs de T1 et T2

pour chaque ligne i1 de T1 :

pour chaque ligne i2 de T2 :

concaténer i1 et i2 dans le résultat

1.7 SQL
- tables jointes -

- T1 INNER JOIN T2 ON condition

  pour chaque ligne i1 de T1 :

  pour chaque ligne i2 de T2 :

  SI i1 et i2 vérifient la condition

  Alors concaténer i1 et i2 dans le résultat

- T1 INNER JOIN T2 USING list_attr
  USING(a,b) = ON(T1.a=T2.a AND T1.b=T2.b)
  a et b n'apparaîtront qu'une fois dans le Résultat

- NATURAL T1 INNER JOIN T2
  comme USING(attrs apparaissant dans T1 et T2)

1.7 SQL
- tables jointes -

- T1 [NATURAL] {LEFT|RIGHT|FULL} OUTER JOIN T2 {ON cond | USING(...)}

  = Résultat d'un INNER JOIN + alpha

  left alpha = Toutes les lignes de T1 qui ne satisfont aucune ligne de T2 sont rajoutées avec des NULLs dans les attrs provenant de T2

  right alpha = Toutes les lignes de T2 qui ne satisfont aucune ligne de T1 sont rajoutées avec des NULLs dans les attrs provenant de T1

  full alpha = left + right

1.7 SQL
- sous-requêtes dans le FROM -

Requêtes SELECT entre parenthèses, dont le résultat est une table dérivé

Exemple:

```
SELECT
   T1.attr1, T1.attr2, alias1.b
FROM
   T1 , T2 , (SELECT a,b,c FROM T3 …) AS alias1
WHERE
   ….
   T2.attr = alias1.b
   ….
```

1.7 SQL
- le 1er Filtre : WHERE condition -

<cond> = exp booléenne, incluant :

- opérateur logiques/relationnels
- exp [NOT] IN (exp, exp,...)
- (exp, exp, ...) [NOT] IN (sous requête)
- exp opérateur ANY (exp, exp,...)
- (exp, exp, ...) opérateur ANY (sous requête)
- exp opérateur ALL (exp, exp,...)
- (exp, exp, ...) opérateur ALL (sous requête)
- EXISTS (sous requête)

1.7 SQL
- le 1er Filtre : exemple -

```
SELECT col1 FROM tab1

WHERE EXISTS (SELECT 1 FROM tab2
                    WHERE col2 = tab1.col2);



SELECT * FROM T WHERE C1 IN (1, 2, 3);



SELECT * FROM T1

WHERE C1 = ANY (SELECT C3 FROM T2

                WHERE C2 = T1.C1 + 10);
```

1.7 SQL
- le 2e Filtre : GROUP BY -

- Utilisée pour regrouper des lignes dans une table qui partagent les mêmes valeurs dans toutes les colonnes listées
- Les colonnes non utilisées dans le groupage ne peuvent pas être référencées sauf dans des expressions agrégats (ex sum, min, max…)

```
SELECT codep, p.nom, (sum(v.qte) * p.prix) AS ventes
FROM produit p NATURAL JOIN vente v
GROUP BY codep, p.nom, p.prix;
```

1.7 SQL
- le 3e Filtre : HAVING -

- Utilisée pour éliminer certains groupes dans une table ayant subit un groupage.
- La condition porte, en général, sur les fonctions agrégats.

```
SELECT codep, p.nom, (sum(v.qte) * p.prix) AS ventes
FROM produit p NATURAL JOIN vente v
GROUP BY codep, p.nom, p.prix
HAVING sum(v.qte) > 100;
```

Références

« An introduction to Database Systems »
C.J.Date, 8th Ed, Addison-Wesley 2004.

« Database Systems »
P. Beynon-Davies, 3rd Ed, Palgrave Macmillan 2004.

« Database System Concepts »
A. Silberschatz, H.F. Korth & S. Sudarshan, 5th Ed, McGraw-Hill 2005.

« Concurrency control and recovery in Database Systems», P. Bernstein, V. Hadzilacos & N. Goodman, Addison-Wesley 1987
(http://research.microsoft.com/~philbe/ccontrol/)