Conception et Exploitation des Systèmes d'Information

Gilles SIMON
Loria, bureau C143
E-mail : gsimon@loria.fr
http://www.loria.fr/~gsimon

ESIAL

Nancy-Université
Université Henri Poincaré

Plan du cours

1. **Conception des SI**
   - 1.1 Dépendances fonctionnelles, normalisation
   - 1.2 Schéma conceptuel des données et passage au schéma logique (rappels)
   - 1.3 Schéma conceptuel des traitements
   - 1.4 Passage au schéma logique des traitements et optimisation
2. **Exploitation des SI**
   - 2.1 Sécurité et cohérence des données
   - 2.2 JDBC
   - 2.3 Object Relational Mapping – Persistance Java
   - 2.4 PHP

1ère partie : Conception des SI

1.1 Dépendances fonctionnelles, normalisation

1.2 Schéma conceptuel des données et passage au schéma logique (rappels)

1.3 Schéma conceptuel des traitements

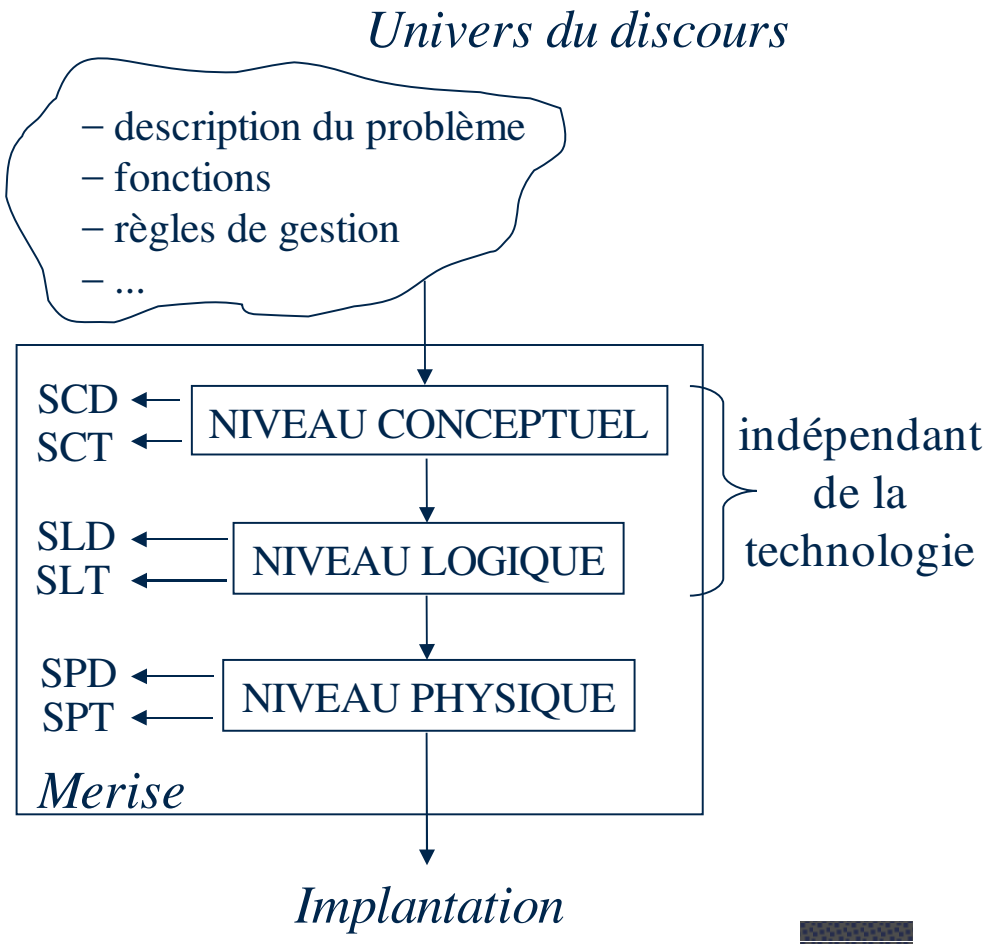
1.4 Passage au schéma logique des traitements et optimisation

La méthode Merise

- Créée en 1977
- Volonté de doter les administrations et les entreprises publiques d'une méthodologie rigoureuse tout en intégrant les aspects nouveaux pour l'époque : *informatique repartie* et *bases de données*
- Propose une approche de conception *séparant l'étude des données de celle des traitements*, en avançant progressivement par niveau : chaque niveau fournit un certains nombre de documents permettant la *synthèse textuelle d'un processus de réflexion*
- Citée en France par 70% des entreprises déclarant utiliser une méthode !

Plusieurs niveaux d'abstraction

- **Niveau conceptuel** : produit la représentation abstraite des données et des traitements (*quoi faire ? avec quelles données ?*)
- **Niveau logique** : transformation et raffinement des résultats du niveau conceptuel, estimation en temps et en volume du futur système (*qui ? ou ?*)
- **Niveau physique** : adapte les résultats du niveau logique aux contraintes du matériel et du logiciel utilisés (*comment ?*)



1.1 Dépendances fonctionnelles, normalisation

Le modèle relationnel : rappel

- Introduit en 1970 par Codd, solides bases théoriques : définir de façon formelle les langages de manipulation associés
- Le modèle relationnel représente l'information dans une collection de relations :
  - une **relation** = une **table** à double entrée
  - chaque **ligne** de la table (appelée **nuplet** ou **tuple**) peut être vue comme un fait décrivant une entité du monde
  - une **colonne** de la table est appelée un **attribut**.
- A chaque relation est associée une *clé* (attribut souligné) qui l'identifie de manière unique

Exemple

- Schéma de la relation Etudiant :

  Etudiant(no,nom,prénom,âge)

- Extension de la relation Etudiant :

attribut

tuple

no	nom	prénom	âge
15	Dupond	Loïc	19
12	Blanc	Michèle	20
1	Noir	Pascal	23
4	Rouge	François	22
8	Tata	Alain	22

Risques

Relation Produit

prod_id	libellé	pu	qte	dep_id	adr	volume
p1	K7	23.5	300	1	Nancy	9000
p1	K7	23.5	500	2	Laxou	6000
p3	Vis	10	900	4	Nancy	2000

- **Redondance** : libellé et pu apparaissent pour chaque instance d'un produit
- **Risque d'introduction d'incohérence** lors de l'insertion d'une nouvelle instance de p1
- **Risque de perte d'information** : suppression du produit p3 ⇒ on perd son libellé, son pu et les information relatives au dépôt 4

Solutions

Deux solutions sont envisageables :

- Transformer les relations dangereuses en plusieurs relations pertinentes → établir les dépendances fonctionnelles, puis normaliser
- Se poser la question *avant* d'écrire les schémas de relations → schéma entités/associations (E/A), puis transformation en relationnel

Les dépendances fonctionnelles

- Un attribut ou une liste d'attributs *Y dépend fonctionnellement* d'un attribut ou d'une liste d'attributs *X* dans une relation *R*, si étant donnée une valeur de *X*, il ne lui est associé qu'une seule valeur de *Y* dans toute instance de *R*. On note $X \xrightarrow{R} Y$ une telle dépendance.
- Exemples :

  $prod\_id \rightarrow libellé$

  $prod\_id \rightarrow pu$

  $dep\_id \rightarrow adr, volume$

  $prod\_id, dep\_id \rightarrow qte$

Propriétés des dépendances fonctionnelles

Soit U la liste des attributs de R, on a alors les propriétés :

(F1) *Réflexivité :* $Y \subseteq X \wedge X \subseteq U \Rightarrow X \xrightarrow{R} Y$

(F2) *Augmentation :* $X \xrightarrow{R} Y \Rightarrow X \cup Z \xrightarrow{R} Y \cup Z$

(F3) *Transitivité :* $X \xrightarrow{R} Y \wedge Y \xrightarrow{R} Z \Rightarrow X \xrightarrow{R} Z$

(F4) *Union :* $X \xrightarrow{R} Y \wedge X \xrightarrow{R} Z \Rightarrow X \xrightarrow{R} Y \cup Z$

(F5) *Pseudo-transitivité :* $X \xrightarrow{R} Y \wedge Y \cup W \xrightarrow{R} Z \Rightarrow X \cup W \xrightarrow{R} Z$

(F6) *Décomposition :* $X \xrightarrow{R} Y \wedge Z \subseteq Y \Rightarrow X \xrightarrow{R} Z$

Typologie des dépendances fonctionnelles

- Une dépendance $X \xrightarrow{R} Y$ est :
  - *triviale* si $Y \subseteq X$
  - *élémentaire* si pour tout $X' \subset X$, la dépendance fonctionnelle $X' \xrightarrow{R} Y$ n'est pas vraie (*Y* ne dépend pas fonctionnellement d'une partie de *X*)
  - *canonique* si sa partie droite ne comporte qu'un seul attribut
  - *directe* si *(i)* elle est élémentaire et si *(ii)* Y ne dépend pas transitivement de X
- Exemples :
  - *prod_id, libelle → libelle ??*
  - *prod_id, libelle → pu* ??
  - *prod_id, dep_id → qte* ??

Clé d'une relation

- Un attribut ou une liste d'attributs X est une *clé* pour la relation $R(X,Y)$ si Y dépend fonctionnellement de X dans R
- X est une *clé minimale* si $X \xrightarrow{R} Y$ est élémentaire
- Si une relation possède plusieurs clés, on en choisit une qui est appelée *clé primaire* (soulignée dans les schémas de relation). Les autres clés sont appelées *clés secondaires*

  Ex : Voiture(immat#, chassis#, type, marque, puissance) admet immat# et chassis# comme clés

Première forme normale

1NF Une relation est dite en *première forme normale* si chacun de ses attributs a un domaine atomique mono-valué

- Exemple :

  Personne(id, nom, les_diplômes) où les_diplômes est l'ensemble des diplômes obtenus par une personne n'est pas en 1NF

  Personne(id, nom) est en 1NF

  Diplôme(id, unDiplome) est en 1NF

Deuxième forme normale

2NF Une relation *R* est dite en *deuxième forme normale* si *(i)* elle est en 1NF et *(ii)* tout attribut n'appartenant pas à une clé ne dépend pas d'une partie de la clé de *R*

- Exemple :

  Stock(prod_id, dep_id, libellé, qte) n'est pas en 2NF car

  $prod\_id, dep\_id \rightarrow qte$
  $prod\_id \rightarrow libellé$

  Stock(prod_id, dep_id, qte) est en 2NF
  Produit(prod_id, libellé) est en 2NF

Troisième forme normale

3NF Une relation est dite en *troisième forme normale* si *(i)* elle est en 2NF et *(ii)* tout attribut n'appartenant pas à une clé ne dépend pas d'un attribut non clé

- Exemple :

Relation Avion

no_avion	constructeur	type	capacité	propriétaire
AH32	Boeing	B747	C1	Air France
FM34	Airbus	A320	C2	British Airways
BA45	Boeing	B747	C1	Egypt Air

n'est pas en 3NF car *type* → *capacité, constructeur* et type n'est pas une clé

Avion(no_avion, type, propriétaire) est en 3NF

Modèle(type, constructeur, capacité) est en 3NF

Algorithme de normalisation par synthèse *simplifié*

0 Ecrire les dépendances fonctionnelles (df)

1 Rendre canoniques et élémentaires les df qui ne le sont pas

2 Représenter les nouvelles df sous forme d'un graphe dont les nœuds sont les attributs impliqués dans les df et les arcs les df elles-mêmes

3 Eliminer les df non directes

4 Partitionner les df en G_1, ..., G_n de façon à ce que toutes les df d'une partition aient la même partie gauche

5 Constituer une relation R_i par G_i, la partie gauche de G_i étant clé de R_i : chaque R_i est alors en 3NF

Exemple

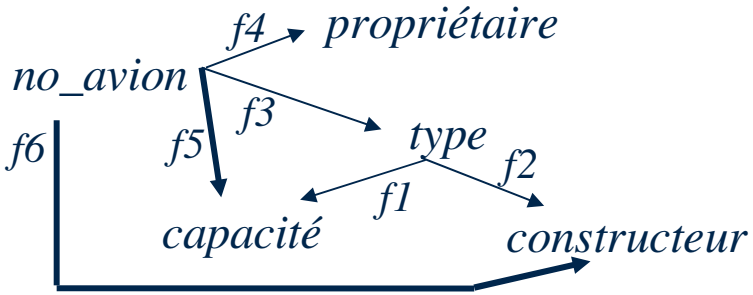
0

type → capacité, constructeur
no_avion → type, propriétaire, capacité, constructeur

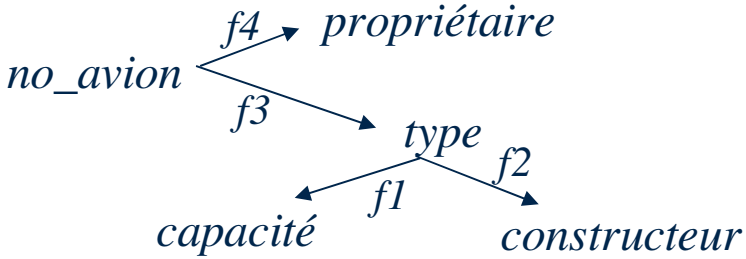
1

type → capacité (f1)
type → constructeur (f2)
no_avion → type (f3)
no_avion → propriétaire (f4)
no_avion → capacité (f5)
no_avion → constructeur (f6)

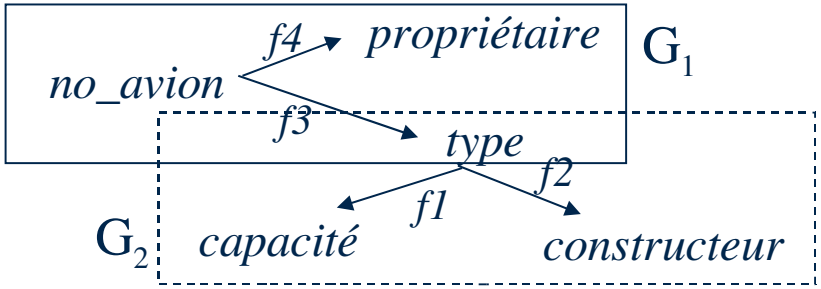
2



3



4



5

Avion(<u>no_avion</u>, type, propriétaire)
Modèle(<u>type</u>, constructeur, capacité)

D'autres formes normales ...

Considérons les df suivantes : code → ville et ville, rue → code

La relation code_postal(ville, rue, code) est-elle en 3NF ???

Oui !!! pourtant, il y a de la redondance :

ville	rue	code
Vandoeuvre	Aiguillettes bld	54506
Vandoeuvre	Gal Leclerc av	54506
Nancy	Saint Jean rue	54000
Nancy	R. Poincare rue	54000

Il existe d'autres formes normales. Ex : Boyce and Codd (BCNF) : "une relation R est en BCNF ssi les seules df élémentaires qu'elle comporte sont celles où une clé détermine un attribut" → code_ville(code, ville) et code_rue(code, rue) sont en BCNF

1.2 Schéma conceptuel des données et passage au schéma logique

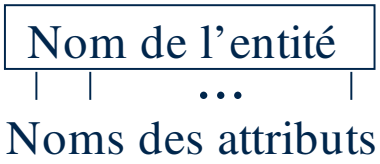
Le modèle Entité-Association

Modèle E/A proposé par Chen en 1976 :

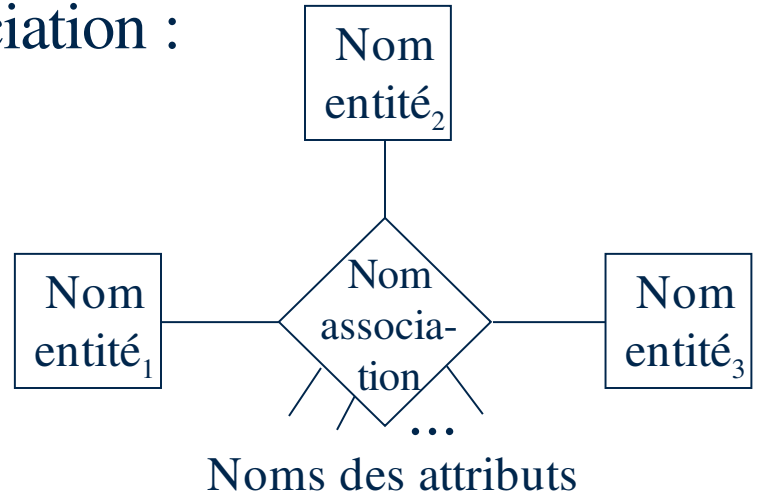
- (Type d') ***entité*** : représentation d'un ensemble d'objets abstraits ou concrets, caractérisée par une liste d'attributs. Un ou plusieurs attributs jouent le rôle de *clé*
- (Type d') ***association*** : permet de décrire les liens "sémantiques" entre des entités, peut être caractérisé par des attributs
- (Type d') ***attribut*** : décrit une propriété attachée soit à une entité, soit à une association. Prend ses valeurs dans un domaine simple (chaîne, entier, réel, ...)

Représentation graphique

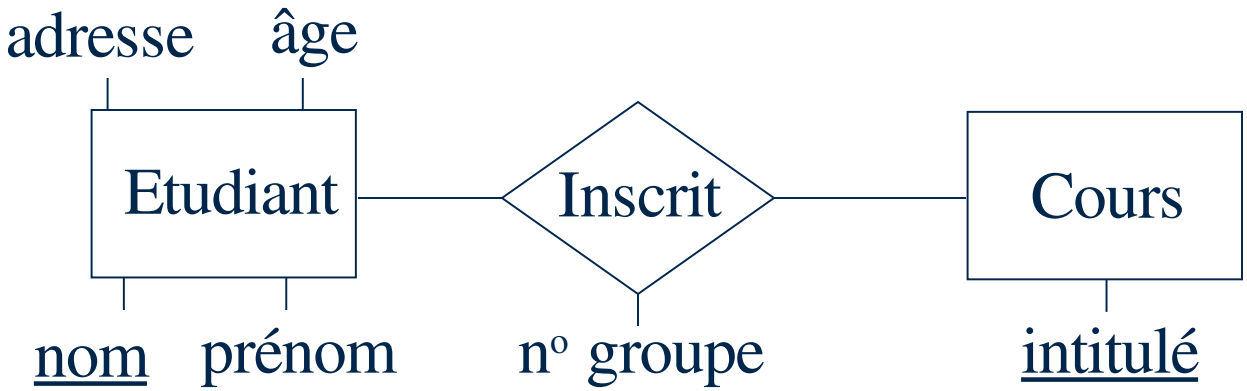
- Type d'entité :



- Type d'association :



Exemple


adresse
âge
Etudiant
Inscrit
Cours
nom
prénom
nº groupe
intitulé

Typologie des associations

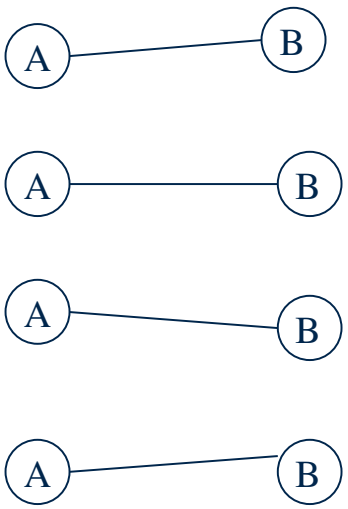
Cardinalité : couple (x,y) x et y exprimant respectivement le nombre minimum et maximum de fois qu'une occurrence du type d'entité peut participer au type d'association

1-1 (*one-to-one*)

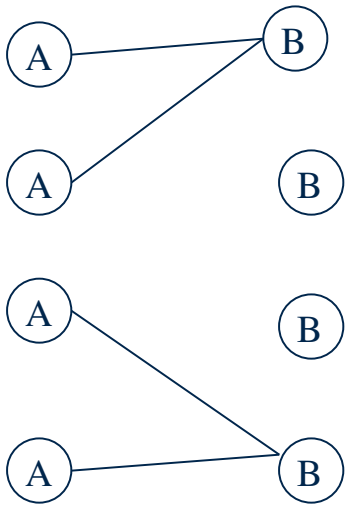
1-n (*one-to-many*)

n-m (*many-to-many*)

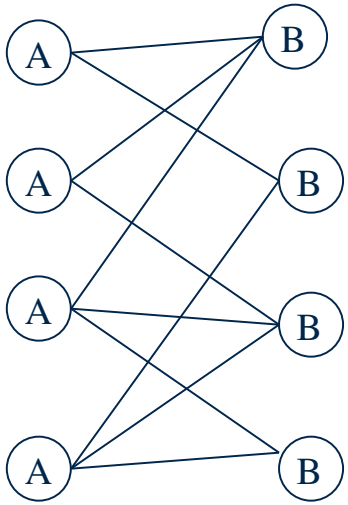
Typologie des associations



One-to-one

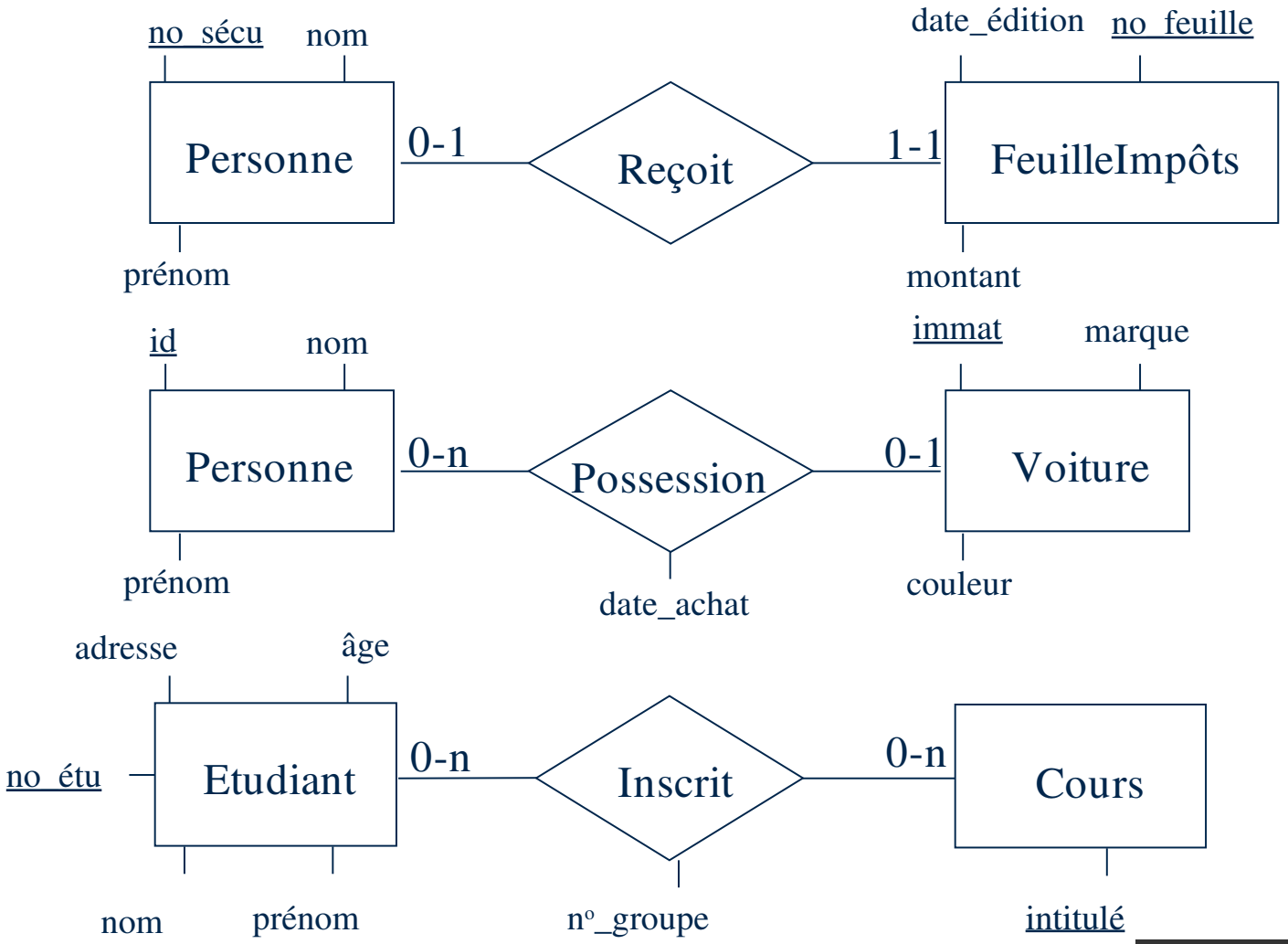


One-to-many



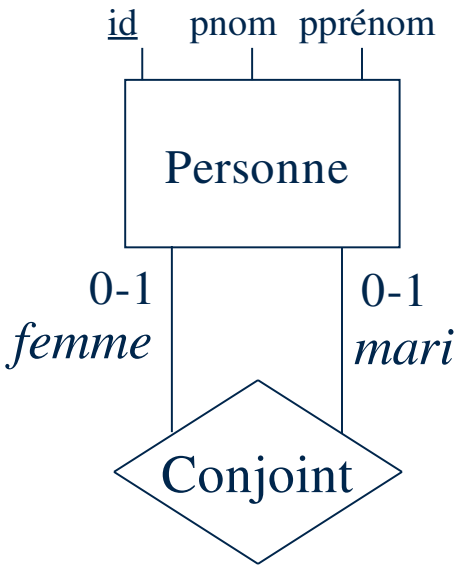
Many-to-many

Exemples


no_sécu
nom
Personne
prénom
0-1
Reçoit
1-1
date_édition
no_feuille
FeuilleImpôts
montant
id
nom
Personne
prénom
0-n
Possession
date_achat
0-1
immat
marque
Voiture
couleur
adresse
âge
no_étu
Etudiant
nom
prénom
0-n
Inscrit
nº_groupe
0-n
Cours
intitulé

Associations réflexives

Possibilité de spécifier le rôle



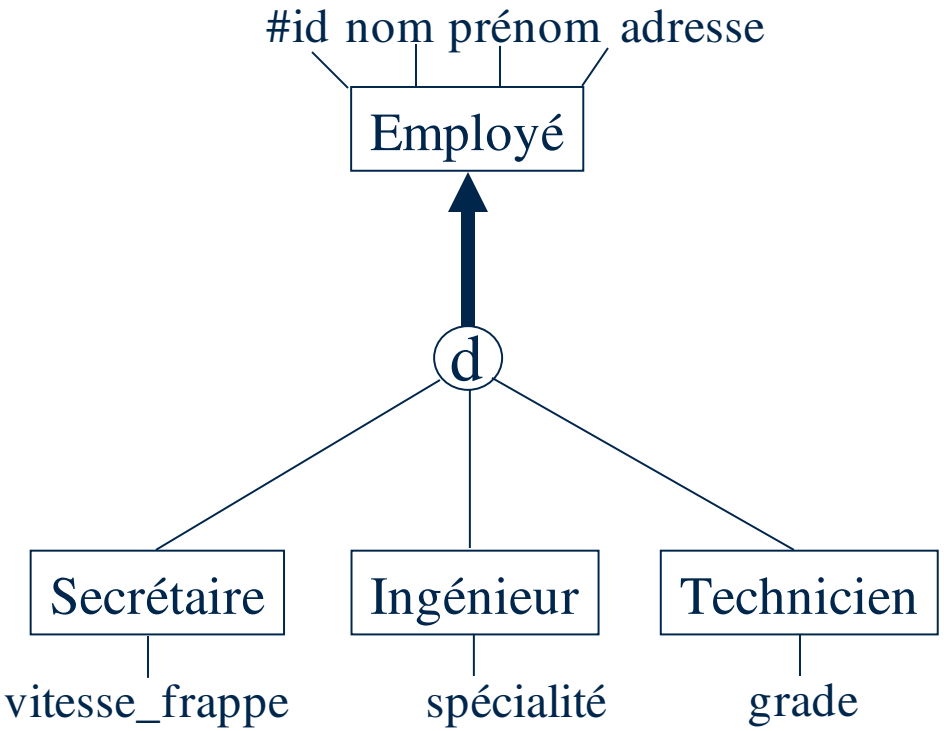
Entités faibles

- Entité faible : entité sans identifiant propre
- N'existe qu'en référence à une autre entité dite *identifiante*
- L'association qui les unit est dite association *identifiante*
- L'entité faible a une cardinalité 1-1 sur son association identifiante
- Exemple :

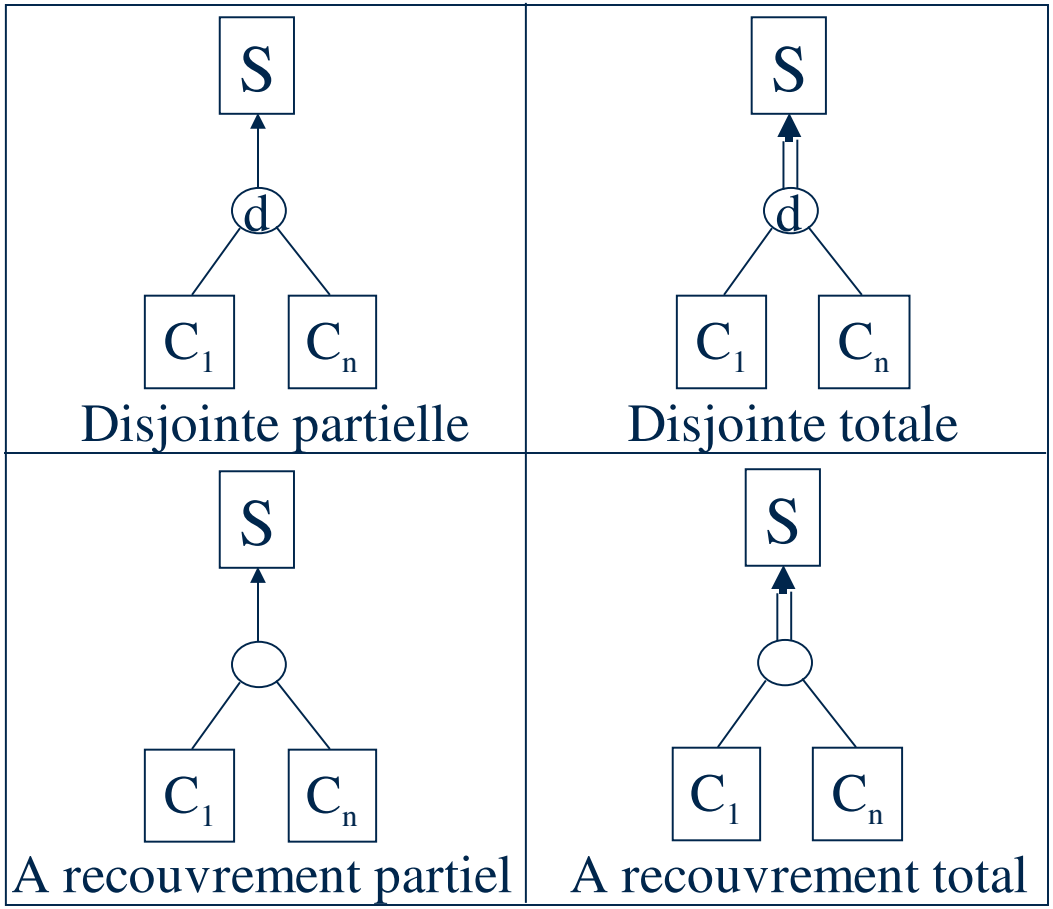
Spécialisations

Définition d'une relation de sous-classe entre une entité *super-classe* et une ou plusieurs entités *sous-classes*

Exemple :



Typologie des spécialisations



ou

Disjointes :

$$\forall i \forall j, i \neq j, \|C_i\| \cap \|C_j\| = \{\ \}$$

A recouvrement :

$$\forall i \forall j, i \neq j, \|C_i\| \cap \|C_j\| \neq \{\ \}$$

ou

Totales :

$$\|S\| = \bigcup_{i=1,n} \|C_i\|$$

Partielles :

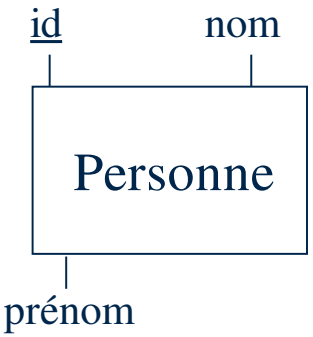
$$\bigcup_{i=1,n} \|C_i\| \subset \|S\|$$

Schéma E/A : Quelques règles à respecter

- Un type d'attribut est caractérisé par un nom et un domaine,
- Les domaines de valeurs des attributs sont *atomiques*
- Les noms des types d'attributs, d'entités et d'associations sont distincts
- Tout type d'entité a au moins un identifiant (une *clé*), qui est un type d'attribut ou une liste de types d'attributs

Passage d'un schéma E/A à un schéma relationnel (1/2)

- Cas des types d'entités :
  - transformer toute entité E en une relation RE
  - les attributs de RE sont les attributs de E
  - la *clé primaire* de RE est un des identifiants de E
- Exemple :



➡ Personne(<u>id</u>, nom, prénom)

Passage d'un schéma E/A à un schéma relationnel (2/2)

1. RAB(Attr (A), Attr(B), Attr(R))

clé(RAB)=clé(A) ou clé(B)

1. RA(Attr(A), Attr(R), clé(RB))
2. RB(Attr(B))

clé(RA)=clé(A) (U clé(B) si A entité faible)

clé(RB)=clé(B)

1. RA(Attr(A))
2. RB(Attr(B))
3. RR(clé(RA), clé(RB), Attr(R))

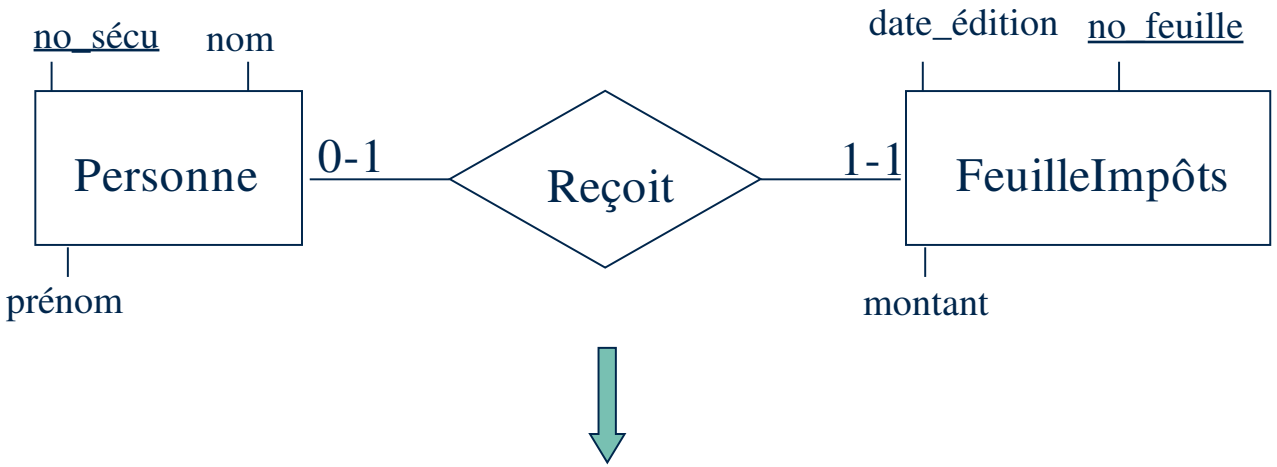
clé(RA)=clé(A)

clé(RB)=clé(B)

clé(RR)=clé(A) U clé(B)

Passage d'un schéma E/A à un schéma relationnel

- Exemple 1



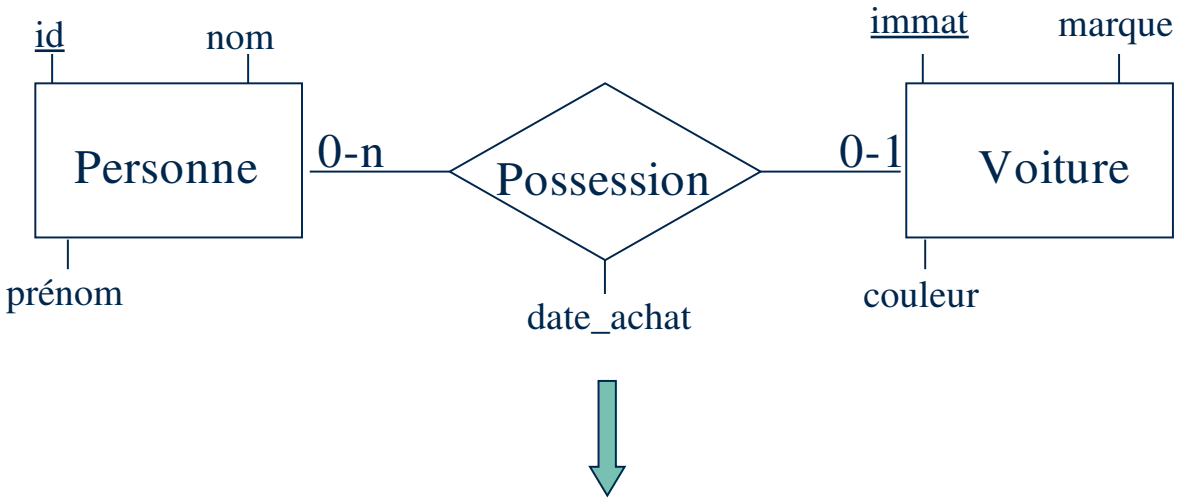
PersonneImpôts(no_sécu, nom, prénom, no_feuille, date_édition, montant)

ou

PersonneImpôts(no_sécu, nom, prénom, no_feuille, date_édition, montant)

Passage d'un schéma E/A à un schéma relationnel

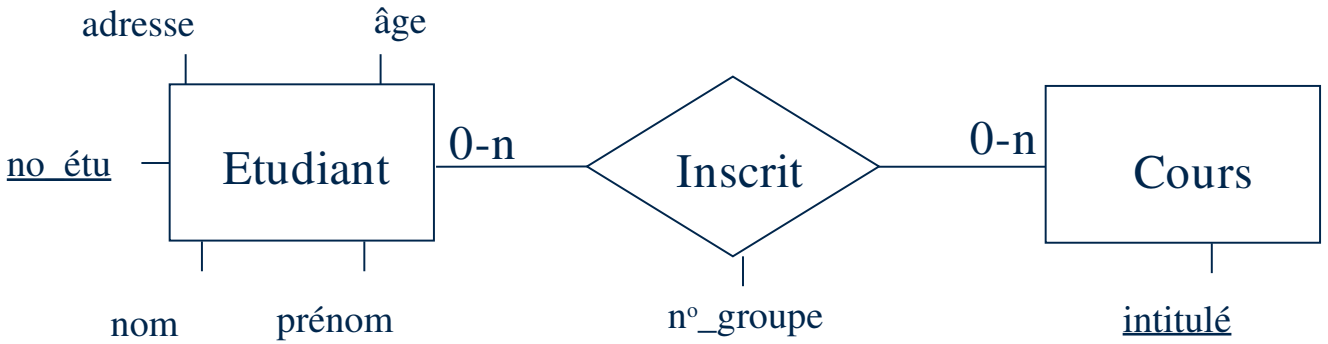
- Exemple 2



Voiture(<u>immat</u>, marque, couleur, id, date_achat)
Personne(<u>id</u>, nom, prénom)

Passage d'un schéma E/A à un schéma relationnel

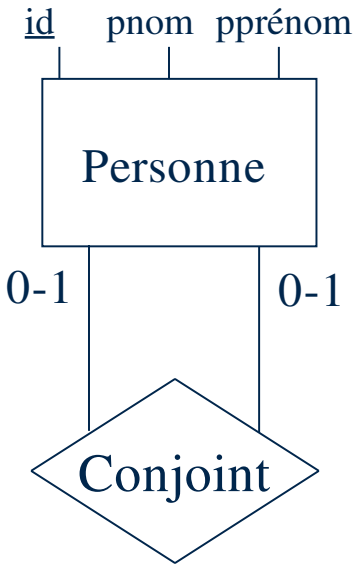
- Exemple 3



Etudiant(no_étu, nom, prénom, age, adresse)
Cours(intitulé)
Inscrit(no_étu, intitulé, no_groupe)

Passage d'un schéma E/A à un schéma relationnel

- Exemple 4

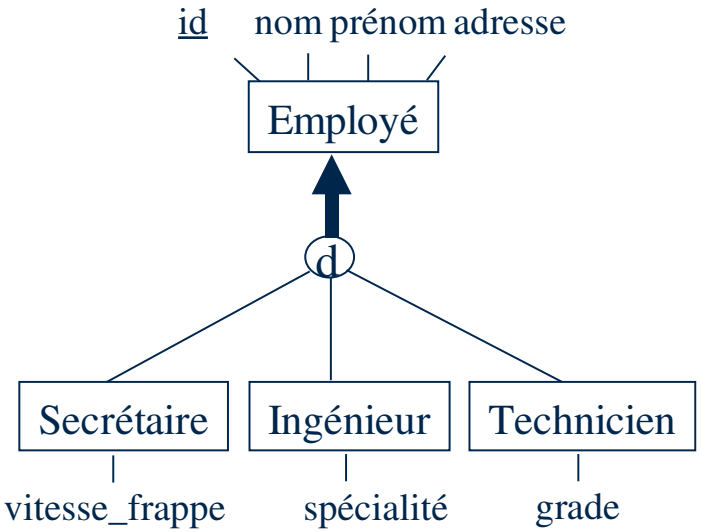


Personne(<u>id</u>, pnom, pprenom, id_conjoint)

où id_conjoint est un *synonyme* de id

Passage d'un schéma E/A à un schéma relationnel

- Cas des spécialisations
  - créer une relation pour chaque entité
  - les attributs de chaque relation sont les attributs de l'entité correspondante + la clé de la super-classe



Employe(id, nom, prenom, adresse)
Secrétaire(id, vitesse_frappe)
Ingénieur(id, spécialité)
Technicien(id, grade)

1.3 Schéma conceptuel des traitements

Le schéma conceptuel des traitements

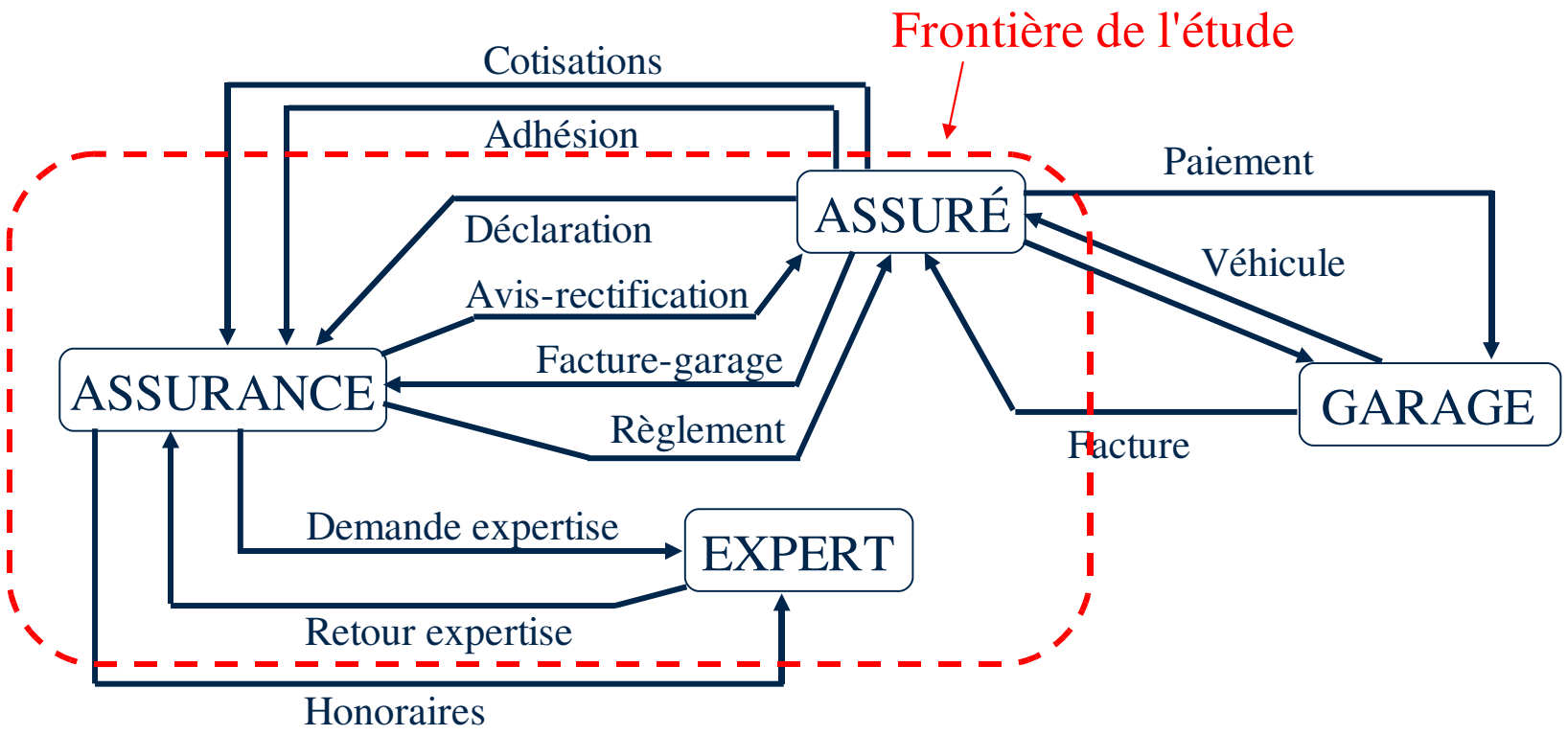
- Un SCT (ou MCT - Modèle ...) est une abstraction des activités du système d'information et de leurs contraintes
- Inspiré des réseaux de pétri
- Processus de conception :
  - Identification des acteurs et des flux d'informations
  - Ordonnancement des flux
  - Elaboration du SCT

Le diagramme de flux

- Sert de base pour fixer la portée et les limites du futur système ou pour le décomposer en sous-systèmes

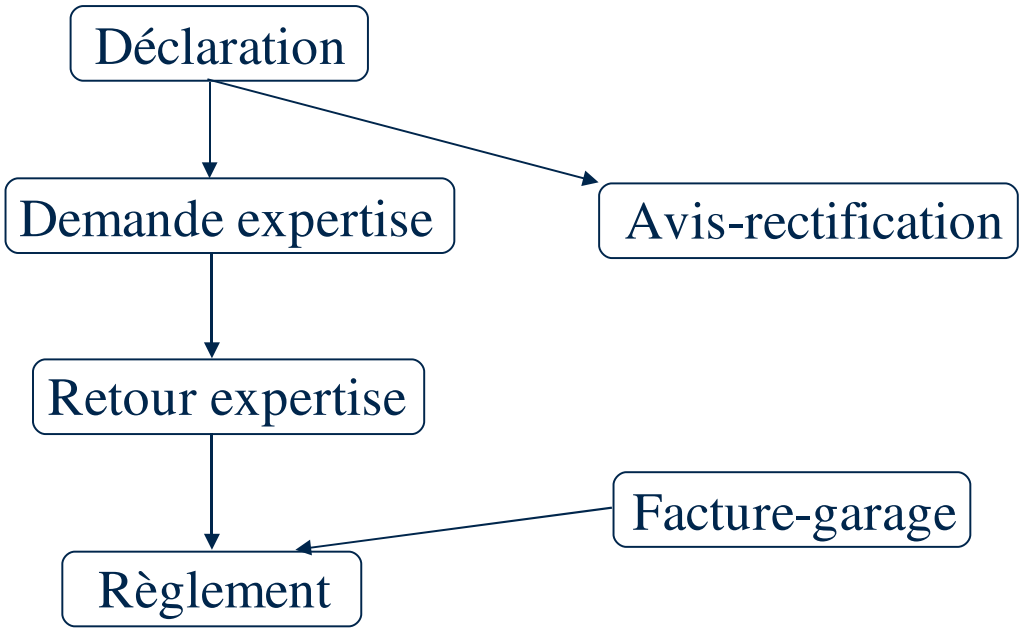
- Se présente sous la forme d'un graphe dont les nœuds sont des acteurs identifiés du SI et les arcs montrent les types d'information circulant entre les acteurs

Exemple de diagramme de flux



Le graphe de précédence

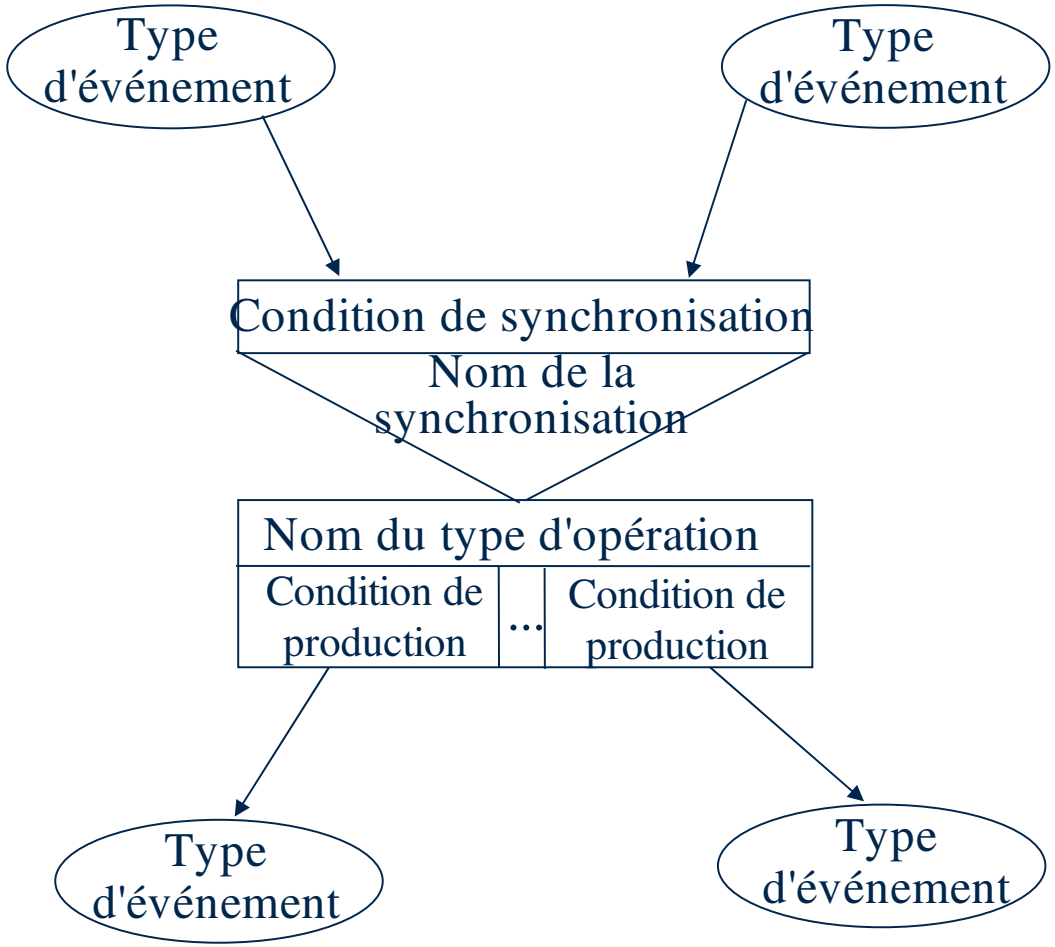
- Montre les dépendances *temporelles* entre les types d'informations
- Exemple :



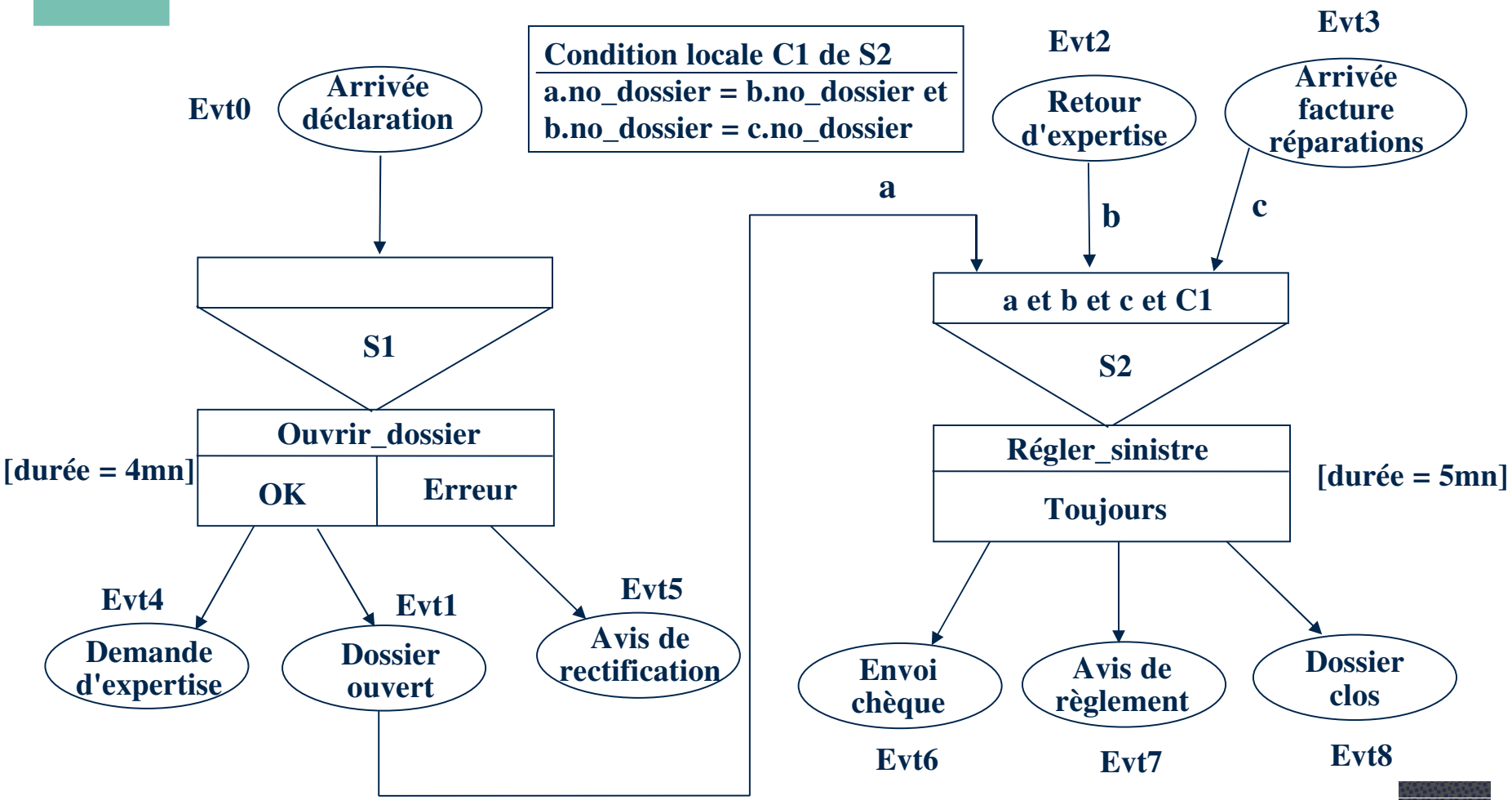
Elaboration du schéma conceptuel des traitements

- En se basant sur le graphe ordonné des flux, on introduit les traitements concernant un ou plusieurs flux
- On décrit le rôle des traitements et les informations nécessaires en entrée et produites en résultat

Représentation graphique du modèle des traitements



Exemple de SCT



Type d'événement

- Description lexicale :
  - *nom* et *message*
  - *identifiant* des occurences
  - *fréquence* d'apparition au cours d'une période donnée
  - *capacité* (nb max d'occurences que le SI peut prendre en compte au cours d'une période)
  - liste des *synchronisations* auxquelles il participe et des opérations qu'il peut déclencher

Exemple de type d'événement

- Evénement *Arrivée déclaration* (Evt0)
  - *message* : informations figurant sur la déclaration
  - *identifiant* : le couple (no de l'assuré, date d'arrivée de la déclaration)
  - *fréquence* : 50 par jour
  - *capacité* : 55 par jour
  - participe à la *synchronisation* S1 du type d'opération Ouvrir_dossier
- La description du message peut être formalisée :

  <Nom_assuré : chaîne, Prénom : chaîne, No_police : chaîne, Date_accident : date ...>

Type d'opération

- Description lexicale :
  - *nom* et *rôle*
  - *durée*
  - type(s) d'événements qui conditionnent son déclenchement (*entrées*)
  - type(s) d'événements produits (*sorties*)
  - si la production des événements est conditionnelle, expliciter la *condition de production* de chaque événement
  - *action* réalisée

Exemple de type d'opération

- Opération *Ouvrir_dossier*
  - *Rôle* : Vérifie une déclaration et initialise l'expertise
  - *Durée* : 10 minutes
  - *Evénements en entrée* : Evt0
  - *Evénements en sortie* : (Evt4 et Evt1) ou Evt5
  - *Action* :

    si déclaration_OK
    alors Ouvrir un dossier de sinistre (Evt1)
    Faire une demande d'expertise de ce dossier (Evt4)
    sinon Renvoyer la déclaration à l'assuré (Evt5)
    fsi

Type de synchronisation

- Description lexicale :
  - *nom*
  - *liste des types d'événements qui participent à la synchronisation*
  - éventuellement, *condition de synchronisation* portant sur les types d'événements
  - *condition locale* : précise, en présence de plusieurs occurrences d'un type d'événements, laquelle choisir
  - *délai de synchronisation* : temps max séparant le moment où la synchronisation est activable et celui où elle est activée
  - *durée limite* : temps max d'attente entre l'arrivée du premier événement et celle du dernier

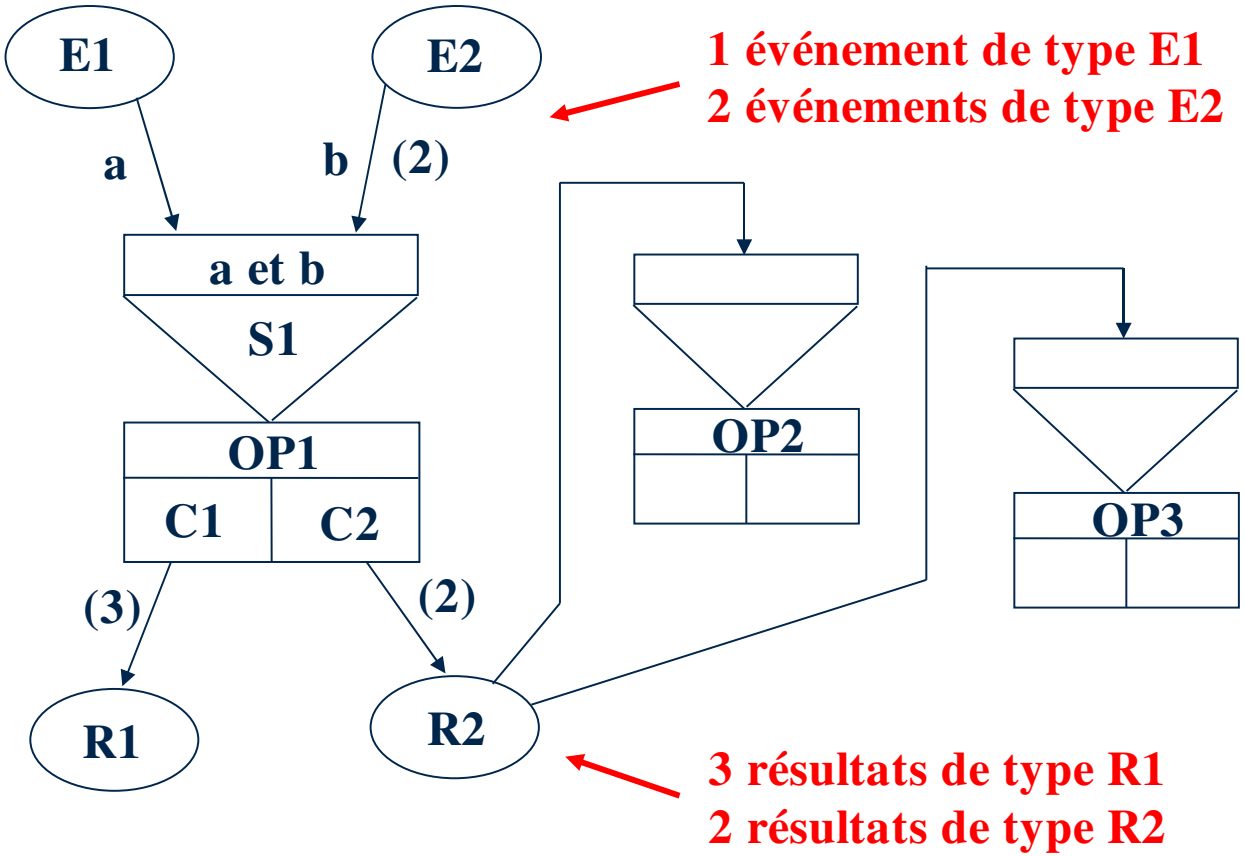
Exemple de type de synchronisation

- Synchronisation S2
  - *Condition* : Evt1 (a) ^ Evt2 (b) ^ Evt3 (c) ^ C1
  - *Condition locale* C1 :

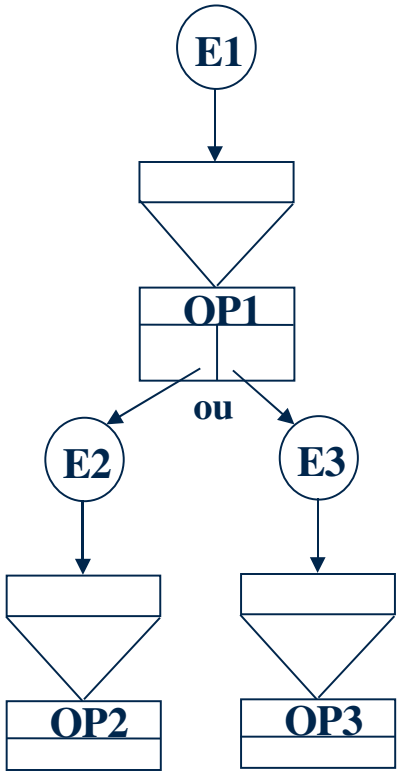
    a.no_dossier = b.no_dossier ^ b.no_dossier = c.no_dossier

    et "premier arrivé premier servi"
  - *Délai de synchronisation* : Trois jours
  - *Durée limite* : douze mois

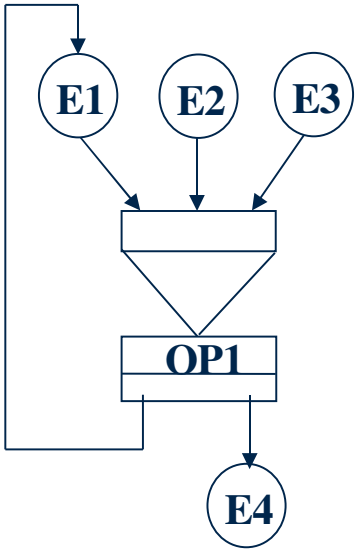
Nombre d'occurences



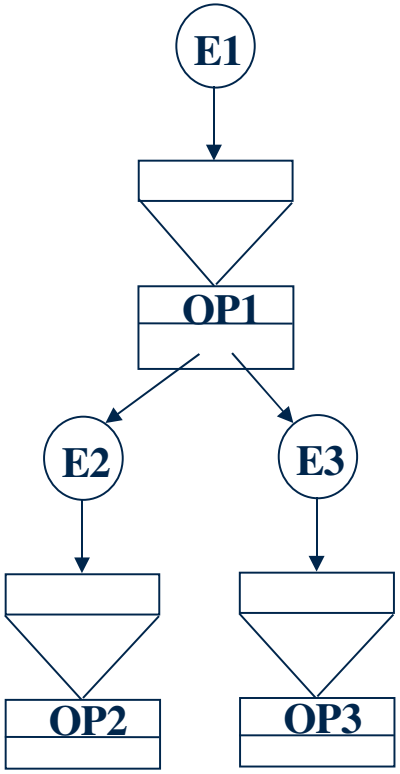
Structures de base d'un SCT



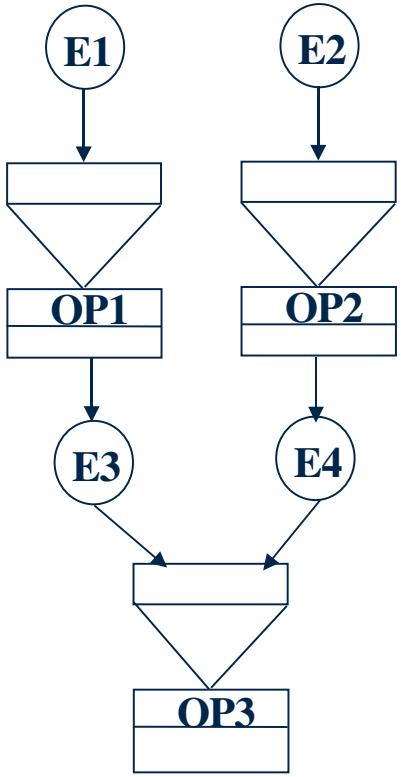
Alternative



Itération



Parallèle divergente



Parallèle convergente

Démarche pratique pour la modélisation conceptuelle

- Elaborer et ordonner un diagramme de flux
- Faire une première ébauche de SCD
- Faire une première ébauche de SCT
- Pour chaque opération du SCT, analyser ses effets sur le SCD
- Modifier ou compléter le SCD
- Modifier ou compléter le SCT
- Itérer sur les trois étapes précédentes

1.4 Passage au schéma logique des traitements et optimisation

Le schéma logique des traitements

- Aboutit à une architecture de déploiement du système, obtenue par raffinement des opérations conceptuelles
- Les étapes de la construction du SLT :
  - décomposer les opérations du SCT en sous-opérations appelées *procédures* ou *fonctions*
  - affecter et localiser chaque procédure
  - détailler l'analyse de chaque procédure
  - définir l'enchaînement des procédures
  - estimer le coût de mise en place de la base
  - essayer de réduire ce coût

Décomposition des opérations

Exemple : l'opération *Ouvrir_dossier* peut être décomposée en les procédures suivantes :

- vérifier la déclaration (assuré connu, circonstances bien décrites ...)
- l'ignorer ou lui affecter un numéro de dossier
- enregistrer les informations nécessaires dans la base
- désigner un expert pour le nouveau dossier
- transmettre le dossier à l'expert

Identification des procédures

Pour chaque procédure sont fournis :

- un nom
- un mode de réalisation (manuelle, automatisée totalement ou partiellement, interactive, différée ...)
- une localisation (où?)
- une affectation (qui?)
- une fréquence d'activation

Exemple

Nom	**No**	**Mode**	**Localisation/ affectation**
Vérifier_déclaration	P1	manuel	Hôtesse
Attribuer_no_dossier	P2	conversationnel	Hôtesse
Enregistrer_dossier	P3	conversationnel	Hôtesse
Désigner_expert	P4	conversationnel	Chef de service
Transmettre_dossier	P5	manuel	Secrétariat du chef de service

Analyse détaillée des procédures

Décrire :

- les événements ou données nécessaires au déclenchement de la procédure et les résultats qu'elle produit
- les traitements effectués et les actions réalisées sur la base : algorithme + algèbre relationnel à partir du SLD
- les supports des données et des résultats (formulaire papier, écrans de dialogue etc.)

Exemple

SLD Assuré(no_ass, nom_ass, adr_ass, tel_ass, no_agence)

Fonction *vérifier_déclaration*

Données

d : déclaration

Début

Si $\sigma_{no\_ass = d.no\_police}(Assuré) = \{\}$

Alors *assuré_inconnu*

Sinon *déclaration_ok*

Fin

Enchaînement des procédures : exemple

Date au plus tôt	Date au plus tard	Enchaînement des procédures
J	J	*Déclaration vérifiéee* → P2 Attribuer_no_dossier
J	J	P3 Enregistrer_dossier
J	J + 2	*Dossier ouvert* → P4 Désigner_Expert
J	J + 3	P5 Transmettre_dossier → *Dossier expédié*

Adaptation des modèles logiques

- Adapter les schémas logiques (SLD et SLT) dans le but de réduire le coût d'implantation de la base
- Facteurs à prendre en compte :
  - volume des données
  - nombre d'accès à la base
  - coût des traitements (supposé négligeable par rapport au coût des lectures - écritures dans la base)

Evaluation des volumes (1/2)

- Taille d'une relation représentant un type d'entité :
  - attribut : nombre de caractères nécessaires à sa représentation
  - n-uplet : somme des tailles de ses types d'attributs
  - relation : produit du nombre d'occurrences du type d'entité par la taille du n-uplet de la relation
- Exemple :

Attribut	Type	Taille
no_ass	entier	10
nom_ass	chaîne	30
adr_ass	chaîne	40
te_ass	chaîne	10
no_agence	entier	10

Si 10000 assurés sont attendus sur une période de deux ans, l'estimation de la taille de la relation est de $10000 \times (10+30+40+10+10) \approx 1\text{Mo}$

Evaluation des volumes (2/2)

- Taille d'une relation représentant un type d'association :

  suppose connu le nombre moyen d'occurrences des types d'entité associés

- Exemple :

  " Un produit est stocké en moyenne dans 3 dépôts "

  $\rightarrow$ *taille de la relation stock* = *taille d'un n-uplet de stock* $\times$ 3 $\times$ *nombre de produits*

Optimisation des volumes

Seul type d'optimisation possible : compression des types d'attributs

→ perte de lisibilité et coût supplémentaire dû au processus de compression et de décompression

Evaluation du coût des traitements

- Dépend du type des opérations de base (recherche, insertion, suppression, modification)
- Processus d'évaluation et d'optimisation :
  1. évaluer le coût de chaque type d'opération (SLD et SLT → fréquence de chaque opération, objets concernés et actions élémentaires à effectuer sur ces objets)
  2. identifier les opérations les plus coûteuses
  3. essayer d'en réduire le coût

Coût de la recherche d'un n-uplet (1/2)

- La recherche d'un n-uplet dans un ensemble de *n* n-uplets coûte :
  - 1 accès si un mécanisme d'accès direct existe
  - en moyenne *n*/2 accès sinon

Coût de la recherche d'un n-uplet (2/2)

- Remarques :
  - coût d'un accès direct < coût d'un accès séquentiel

    → *créer des index*
  - coût d'une recherche dans un ensemble ordonné < coût d'une recherche dans un ensemble non ordonné

    → *ordonner les instances*
  - coût d'une lecture < coût de plusieurs lectures

    → *fusionner des relations, introduire de la redondance ou grouper physiquement des occurrences (clustering)*

Coût de l'ajout d'un n-uplet

- Insertion :
  - écriture dans la base
  - mise à jour éventuelle des index existant sur la relation concernée
- Coût :
  - 1 écriture si la relation n'est pas ordonnée
  - *n*/2 lectures en moyenne pour la recherche du point d'insertion si la relation est ordonnée

Coût de la modification d'un n-uplet

- Modification :
  - recherche dans la base
  - modification en mémoire centrale
  - réécriture dans la base
- Coût :
  - coût d'une recherche
  - coût d'un ajout
  - (éventuellement) coût du maintien d'ordre
  - (éventuellement) coût de mise à jour d'index
  - (éventuellement) coût de mise à jour de données redondantes

Coût de la suppression d'un n-uplet

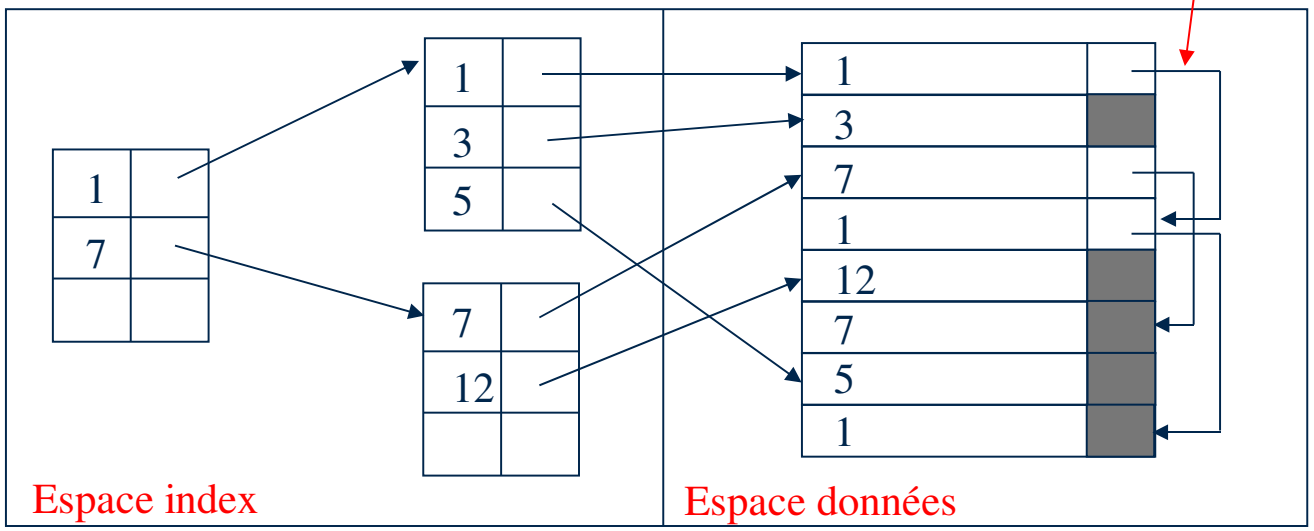
- Suppression :
  - recherche du tuple à supprimer
  - mise-à-jour éventuelle d'index
  - autres suppressions, dans le cas de données redondantes
- Coût :
  - coût d'une recherche
  - (éventuellement) coût de la maintenance des index
  - (éventuellement) coût de la suppression des données redondantes

Optimisation des traitements

- Index et critères d'ordre
- Redondance et dénormalisation de relations
- Ajout de nouveaux types d'attributs
- Fusion de relations
- Fragmentation verticale de relations

Rappel sur les index

- Index : structure de données qui associe à une valeur d'un attribut, appelé clé de l'index, la ou les adresses des tuples contenant cette valeur
- Exemple :



Choix des index et des critères d'ordre

- Opérations d'interrogation :

  attributs de sélection et de jointure

- Opérations de mise à jour :

  attributs de sélection

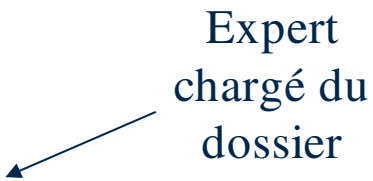
  pas les attributs à modifier car cela entraînerait un coût supplémentaire pour la maintenance

Redondance et dénormalisation de relations

- Exemple :

  Expert(no_exp, nom_exp, ...)

  Sinistre(no_dossier, ..., no_exp)

  Si le nom de l'expert est accédé à chaque référence à un sinistre →
  Sinistre(no_dossier, ..., no_exp, **nom_exp**)

  Viole la 3NF de Sinistre, mais permet de faire l'économie de l'opération de jointure pour obtenir le nom

→ accroissement de l'espace de stockage et de l'activité en mise à jour

Ajout de nouveaux types d'attributs

- Exemple :

  " L'expert désigné pour un dossier de sinistre est celui qui a le moins de dossiers en cours d'instruction "

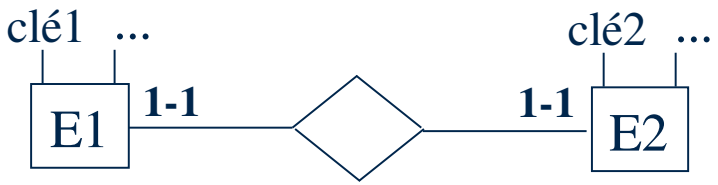
  → nécessite le parcours de la relation sinistre avec comptage et recherche du no_exp ayant le plus petit nombre de dossiers puis accès à la relation Expert pour connaître ses coordonnées

  → Expert(no_exp, nom_exp, ..., **nb_dossiers-en_cours**)

→ Attention à la cohérence de la base :

incrémenter nb_dossiers-en_cours à chaque fois que l'expert est désigné, et décrémenter quand le dossier est clos

Fusion de relations



RE1(clé1, ...)

RE2(clé2, ...)

→ si jointures très fréquentes de RE1 et RE2, fusionner en une seule relation

Fragmentation verticale de relations

- Quand une relation comporte un grand nombre d'attributs, dont seul un sous-ensemble est fréquemment utilisé, on peut la décomposer en deux relations
- Exemple :

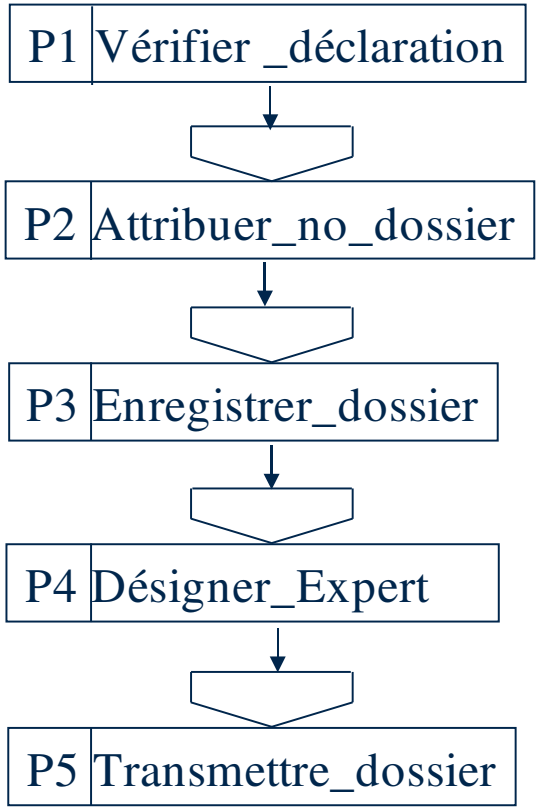
R(cléR, utilisé1, utilisé2, peu_utilisé1, peu_utilisé2, peu_utilisé3)

R1(cléR, utilisé1, utilisé2)

R2(cléR, peu_utilisé1, peu_utilisé2, peu_utilisé3)

Exemple : coût de l'opération *Ouvrir_dossier*

Pour 10000 dossiers et 20 experts :



Recherche de l'assuré → 10000/2 accès

Accès au dernier numéro de dossier affecté → 1 accès

Ecriture du nouveau dossier → 1 accès

Recherche de l'expert → 20 / 2 accès
Ajout d'un dossier à la relation Sinistre → 1 accès

Total : 5013 accès × 50 sinistres par jour × 260 jours ouvrés → coût annuel de **65.10⁶** accès

Coût de l'opération *Ouvrir_dossier* optimisée

- Si on décide de :
  - conserver en mémoire le dernier numéro de dossier pendant toute une journée de travail
  - indexer les assurés sur leur numéro de police
  - indexer les experts sur le nombre de dossiers en cours avec ordre

→ Nouveau coût :

260 jours × (1 accès au dernier numéro de dossier +
50 sinistres ×
(1 pour la recherche de l'assuré
+ 1 pour l'écriture du nouveau dossier
+ 1 pour la recherche de l'expert
+ 1 pour l'ajout d'un sinistre)) = **52.10^3** accès / an

2ème partie : Exploitation des SI

2.1 Sécurité et cohérence des données

2.2 JDBC

2.3 Object Relational Mapping – Persistance Java

2.4 PHP

2.1 Sécurité et cohérence des données

Sécurité et cohérence des données

- Confidentialité : empêcher l'accès à la BD ou à une de ses parties par des personnes non autorisées
- Intégrité : préserver la BD dans un état cohérent vis-à-vis des contraintes spécifiées
- Sécurité de fonctionnement : gérer les accès concurrents

Confidentialité

- Tout utilisateur d'un SGBD doit posséder un compte et un mot de passe
- Utilisateur privilégié : l'administrateur
  - a le droit de créer des comptes
  - d'allouer des privilèges d'accès
- Le système garde une trace de toutes les actions effectuées lors d'une session (*audit*)

Deux niveaux de privilèges

- Privilèges de niveau compte :

  définissent pour un compte, le type de commandes autorisées sur des types d'objets (délivrés par l'administrateur) : **`CREATE {TABLE|VIEW}, DROP {TABLE|VIEW}, …`**

- Privilèges de niveau objet :

  spécifient le type des commandes autorisées sur une base, une relation, une vue ou une procédure stockée (délivrés par le propriétaire de l'objet) ; doivent être compatibles avec les privilèges détenus au niveau compte

- Propagation de privilèges

  Possibilité de transmettre ses privilèges sur une relation à un autre compte

  Rôle : ensemble nommé de privilèges

Privilèges Oracle (1/2)

- Gestion des privilèges de niveau compte

```
GRANT {privilege_systeme | rôle} [,…] TO {PUBLIC|
   nom_utilisateur|rôle} [,…] [IDENTIFIED by
   mot_de_passe][WITH ADMIN OPTION]

(REVOKE … FROM)
```

liste_de_commandes : **CREATE TABLE, CREATE ANY TABLE, SELECT TABLE, SELECT ANY TABLE, ...**

Privilèges Oracle (2/2)

- Gestion des privilèges de niveau objet

```
GRANT privilège_objet [,…] ON [schéma].objet TO
  {PUBLIC|nom_utilisateur|rôle} [,…] [WITH ADMIN
  OPTION]
```

Exemples

Administrateur :

```
GRANT CREATE TABLE TO ELLE

REVOKE CREATE TABLE FROM ELLE
```

Utilisateur u1 :

```
GRANT SELECT ON u1.Depot TO u2

REVOKE SELECT ON u1.Depot FROM u2
```

Rôles Oracle

- Exemples de rôles :
  - Rôle dba: tous les privilèges (*data base administrateur*)
  - Rôle connect: ouvrir une session, créer des tables, etc.
- Créer et attribuer un rôle :

```
create role un_role
grant create session, create table to un_role
grant un_role to gilles
```

- Supprimer un rôle :

```
revoke un_role from gilles
```

Les contraintes d'intégrité (CI)

- Contraintes que doivent vérifier les données pour être considérées comme des données de qualité
- Une base de données est dite *cohérente* si ses contraintes d'intégrité sont satisfaites

Typologie des contraintes d'intégrité (1/2)

- **Les contraintes individuelles** : concernent un type de donnée (contraintes de domaine ou de valeur obligatoire)

- **Les contraintes intra-relation** : portent sur les valeurs des attributs (unicité de valeur, clé, cardinalité "*le dépôt 30 ne stocke que les produits p1 et p2*", dépendances entre des valeurs de différents attributs "*les produits p4 et p5 ne doivent pas être stockés dans un même dépôt*")

Typologie des contraintes d'intégrité (2/2)

- **Les contraintes inter-relations** : dépendances référentielles ou existentielles ("*tout numéro de produit figurant dans la relation stock doit être présent dans la relation produit*")
- **Les contraintes dynamiques** : les valeurs des données dépendent de l'état de la base ou d'un des états précédents ("*le salaire d'un employé ne peut pas diminuer*")
- **Les contraintes temporelles** : Ex "*toutes les commandes de la semaine courante doivent avoir été facturées avant le jour ouvrable de la semaine suivante*"

Création d'une table

```
CREATE TABLE Depot
   (dep#       INT NOT NULL,
    adr        VARCHAR(50),
    capacite   INT,
    PRIMARY KEY dep#,
    CHECK (capacite>1000))
CREATE TABLE Stock
   (prod#      INT,
    dep#       INT CHECK (dep# BETWEEN 10 AND 50),
    qte        INT DEFAULT 0 CONSTRAINT ctr_qte,
    PRIMARY KEY (prod#, dep#),
    FOREIGN KEY (prod#) REFERENCES Produit(prod#), /* tout tuple
    de Stock correspond a un produit existant dans Produit */
    FOREIGN KEY (dep#) REFERENCES Depot(dep#),
    CHECK (qte>=0))
```

Les déclencheurs (*triggers*)

- Trigger : procédure associée à une relation qui entre en action quand un type d'événement survient
- Créé par le propriétaire de la relation ou un utilisateur ayant le privilège CREATE TRIGGER.
- Les triggers permettent de :
  - maintenir des ci complexes, ce que ne permettent pas les contraintes déclaratives spécifiées lors de la création d'une table
  - publier des informations concernant divers événements,
  - etc…

Triggers Oracle (1/4)

- Triggers after ou before
- Triggers de niveau instruction ou de niveau ligne
- Syntaxe :

```
CREATE [OR REPLACE] TRIGGER nom_trigger
{BEFORE | AFTER | INSTEAD OF} evenement
[clause_referencing] [WHEN condition]
[FOR EACH ROW]
[DECLARE …]
BEGIN
…
[EXCEPTION …]
END [nom_trigger];
```

Triggers Oracle (2/4)

Algorithme de contrôle de l'exécution d'une instruction LMD déclenchante :

1. exécute les déclencheurs BEFORE de niveau instruction, s'il en existe
2. pour chaque ligne affectée par l'instruction :
   a) exécute les déclencheurs BEFORE de niveau ligne, s'il en existe
   b) exécution de l'instruction
   c) exécute les déclencheurs AFTER de niveau ligne, s'il en existe
3. Exécute les déclencheurs AFTER de niveau instruction, s'il en existe

Triggers Oracle (3/4)

- Pour les déclencheurs de niveau ligne, possibilité d'accéder aux données de la ligne en cours de traitement au moyen de deux identifiants de corrélation :
  - :OLD : valeurs d'origine de la ligne avant le traitement
  - :NEW : valeurs qui seront insérées ou remplaceront celles d'origine au terme de l'instruction
- :OLD est non défini pour les instructions INSERT
- :NEW est nom défini pour les instruction DELETE

Triggers Oracle (4/4)

- Exemple :

```
CREATE SEQUENCE inc START WITH 8 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER categoriesTrig
BEFORE INSERT ON categories
FOR EACH ROW
BEGIN
      SELECT inc.NEXTVAL INTO :NEW.id_categorie;
END categoriesTrig

INSERT INTO CATEGORIES (nom_categorie, description) VALUES ('new cat', 'desc cat');

SELECT * FROM categories WHERE id_categorie = 9;

id_categorie   nom_categorie                  description
--------------  ---------------------------------  ------------
 9              new cat                        desc cat
```

Sécurité de fonctionnement et accès concurrents

- Une base de données est accédée par plusieurs utilisateurs ou programmes en même temps → les SGBD comportent un sous-système de gestion des accès concurrents
- Techniques qui s'inspirent de celles développées pour les systèmes d'exploitation (verrouillage, exclusion mutuelle)
- *Transaction :* tout programme qui accède à la base ou qui en modifie le contenu (suite de lectures/écritures)

Problèmes liés aux transactions : non atomicité

Virement d'un montant *m* d'un compte C_1 vers un compte C_2 :

Début
$x \leftarrow Lire(C_1)$
$x \leftarrow x - m$
$C_1 \leftarrow Ecrire(x)$
$y \leftarrow Lire(C_2)$
$y \leftarrow y + m$
$C_2 \leftarrow Ecrire(y)$
Fin

Pb : Si le programme s'arrête avant l'écriture de y dans C_2, la base entre dans un état incohérent

→ Transaction = **unité atomique d'actions**

(si la transaction n'a pas pu exécuter toutes ses actions, la base est remise dans l'état où elle se trouvait avant le début de la transaction)

Problèmes liés aux transactions : non isolation

T1 inscrit un débit *N* sur un solde *X* et *T2* inscrit un crédit *M* sur *X* :

T1
$x \leftarrow Lire(X)$
$x \leftarrow x - N$
$X \leftarrow Ecrire(x)$

T2
$y \leftarrow Lire(X)$
$y \leftarrow y + M$
$X \leftarrow Ecrire(y)$

Ex : $X = 100$, $N = 10$, $M = 50$

Résultat attendu : 100-10+50 = **140**

Actions	*X*	*x*	*y*
T1: $x \leftarrow Lire(X)$	*100*	*100*	
T1: $x \leftarrow x - N$	*100*	*90*	
T2: $y \leftarrow Lire(X)$	*100*	*90*	*100*
T2: $y \leftarrow y + M$	*100*	*90*	*150*
T1: $X \leftarrow Ecrire(x)$	*90*	*90*	*150*
T2: $X \leftarrow Ecrire(y)$	***150***	*90*	*150*

Pb : l'opération de débit est perdue !

→ Il faudrait que T1 travaille en **isolation**, c'est-à-dire sans interférence avec d'autres transactions, jusqu'à sa terminaison

Problèmes liés aux transactions : non cohérence

Ex : on impose la CI (A = B)

Actions de T1
$t11 : A_{T1} \leftarrow Lire(A)$
$t12 : A_{T1} \leftarrow A_{T1} + 1$
$t13 : A \leftarrow Ecrire(A_{T1})$
$t14 : B_{T1} \leftarrow Lire(B)$
$t15 : B_{T1} \leftarrow B_{T1} + 1$
$t16 : B \leftarrow Ecrire(B_{T1})$

Actions de T2
$t21 : A_{T2} \leftarrow Lire(A)$
$t22 : A_{T2} \leftarrow A_{T2} \times 2$
$t23 : A \leftarrow Ecrire(A_{T2})$
$t24 : B_{T2} \leftarrow Lire(B)$
$t25 : B_{T2} \leftarrow B_{T2} \times 2$
$t26 : B \leftarrow Ecrire(B_{T2})$

Pb : chaque transaction satisfait la contrainte, mais leur exécution concurrente peut la violer

Ex : <t21; t22; t11; t12; t23; t13; t14; t15; t16; t24; t25; t26>

Problèmes liés aux transactions : non permanence des modifications

- Non répétabilité des lectures

Transaction T1	*Transaction T2*
a ← Lire(A)	
	b ← Lire(A)
	Imprimer(b)
a ← a + 100	
A ← Ecrire(a)	
	b ← Lire(A)
	Imprimer(b)

Pb : les impressions de T2 produiront des valeurs différentes

Problèmes liés aux transactions : non permanence des modifications

- Tuples "fantômes"

Transaction T1	Transaction T2
S ← Lire(A), A = {a | a < 4000}	V ← Lire(A), A={a | a < 4000} Imprimer(V)
Pour chaque s, s ∈ S : a = 5000 A ← Ecrire(S)	
	V ← Lire(A), A={a | a < 4000} Imprimer(V)

Pb : La deuxième impression de V rend un ensemble vide

Les modifications des données par T1 ne sont pas répercutées dans la base avant leur utilisation par T2

Plusieurs outils pour résoudre ces problèmes

- *Le sous-système d'intégrité* : permet de détecter les incohérences prévues par le programmeur (CI)
- *Le sous-système de reprise* : permet d'assurer l'atomicité des transactions et la cohérence de la base en cas d'accident
- *Le sous-système de contrôle de la concurrence* : permet de contrôler les accès concurrents aux données par la technique des "verrous"

Atomicité des transactions : politique du "tout ou rien"

```
BEGIN TRAN MaTransaction
        Action1
        ...
        Action n
Si toutes les actions se sont bien passées
Alors COMMIT TRAN
Sinon ROLLBACK TRAN
```

→ Valide la transaction

→ Annule les effets de la transaction

Utilisation de points de sauvegarde

```
BEGIN TRAN MaTransaction
        {Séquence 1 d'actions}
SAVE TRAN Point1
        {Séquence 2 d'actions}
SAVE TRAN Point2
        {Séquence 3 d'actions}
ROLLBACK TRAN Point2
        {Séquence 4 d'actions}
ROLLBACK TRAN
```

→ Annule les effets de *Séquence 3*

→ Annule les effets de toutes les séquences

Transactions imbriquées

```
BEGIN TRAN
        ...
        BEGIN TRAN
                ...
                BEGIN TRAN
                        ...
                COMMIT/ROLLBACK TRAN
                ...
        COMMIT/ROLLBACK TRAN
        ...
COMMIT/ROLLBACK TRAN
```

Il y a annulation des effets de *toutes* les transactions si *une* des transactions filles ou si la transaction mère échoue

Le journal des transactions

- Pour pouvoir procéder à la reprise en cas d'accident, le système garde une trace (*journal* ou *log*) des opérations (transactions) effectuées sur la base

- Le journal est conservé sur un support non volatile (disque, bande) et doit être archivé périodiquement

Le journal des transactions

- Pour chaque transaction T, le journal comporte :
  - *<début_transaction, identification de T>*
  - *<écriture, identification de T, donnée concernée, ancienne valeur, nouvelle valeur>*
  - *<lecture, identification de T, donnée concernée>*
  - *<COMMIT, identification de T>*
- Ainsi si un accident se produit, le système peut
  - annuler (UNDO) les effets des transactions non validées
  - refaire (REDO) les actions des transactions validées

Reprise en cas d'accident

- Reprise (*recovery*) : reconstruire un état cohérent de la base à partir du passé, cet état devant être le plus proche possible de l'instant où s'est produit l'incident → utilisation du journal des transactions

- La stratégie de reprise dépend de la gravité de l'incident qui la provoque :
  - *Reprise à froid* : pour des dégâts importants, on recharge une sauvegarde de la base et on ré-exécute (REDO) les transactions marquées valides dans le journal, depuis la date de la sauvegarde
  - *Reprise à chaud* : pour des dégâts moins importants, on défait (UNDO) les actions des transactions non validées

Contrôle de la concurrence et plan d'exécution

Un *plan d'exécution P* de *n* transactions est un ordre sur les opérations des transactions tel que pour toute transaction T_k participant à *P*, si une opération O_i précède une opération O_j dans T_k, alors O_i précède aussi O_j dans *P*

→ **L'objectif du contrôle de la concurrence est de n'autoriser que les plans d'exécution corrects**

Exemple : problème du transparent 62

- <t11; t12; t13; t21; t22; t23; t14; t15; t16; t24; t25; t26> est un plan d'exécution correct
- <t21; t22; t11; t12; t23; t13; t14; t15; t16; t24; t25; t26> est un plan d'exécution incorrect

Plan d'exécution sérialisable

- Un plan d'exécution P des transactions T_1, ..., T_n est une *succession* s'il existe une permutation $\prod$ de (1, ..., n) telle que $P = <T_{\prod(1)} ; ...; T_{\prod(n)}>$
- Une exécution des transactions T_1, ..., T_n est *sérialisable* ssi elle donne, pour chaque T_i, le même résultat qu'une succession

→ **Le problème revient à ce que ne soient autorisées que des exécutions sérialisables**

Contrôle de la concurrence : la technique du verrouillage

Granule = unité d'accès (BD, relation, tuple, attribut ...)

- **Verrou binaire** : un granule est accessible ou inaccessible
- **Verrou partagé et verrou exclusif** : en *lecture* un verrouillage en mode partagé (*share*) permet des accès multiples à un granule. Par contre, en *écriture*, il est nécessaire de verrouiller le granule en mode exclusif (*exclusive*)
- **Verrou d'intention** ou verrou de mise à jour (*update*) : verrou de lecture avec *intention* d'écriture

Compatibilité des verrous :

verrous demandés sur le même granule

verrous apposés sur un granule

	Share	Exclusive	Update
Share	oui	non	non
Exclusive	non	non	non
Update	oui	non	non

Les quatre niveaux d'isolation SQL-92 (1/2)

- Niveau 0 (**`READ UNCOMMITTED`**)

  Une transaction T peut lire des objets modifiés par une autre transaction, pas de verrou en mode lecture

  → risque de lectures impropres, lectures non reproductibles et tuples fantômes

- Niveau 1 (**`READ COMMITED`**)

  = niveau 0, sauf : T lit uniquement les mises-à-jour des transaction validées (verrou *exclusif* en écriture)

  → plus de risque de lectures impropres

Les quatre niveaux d'isolation SQL-92 (2/2)

- Niveau 2 (**`REPEATABLE READ`**)

  = niveau 1, sauf : aucun objet lu par T ne peut être modifié par une autre transaction (verrou *partagé* en lecture)

  → plus de risque de lectures non reproductibles

- Niveau 3 (**`SERIALIZABLE`**)

  = niveau 2, sauf : possibilité de poser des verrous sur un ensemble d'objets

  → plus de risque de tuples fantômes

Application au SGBD Sybase

- Sybase supporte les niveaux d'isolation 0, 1, 2 et 3
- Pour une transaction ou une session, un utilisateur peut choisir un des niveaux :

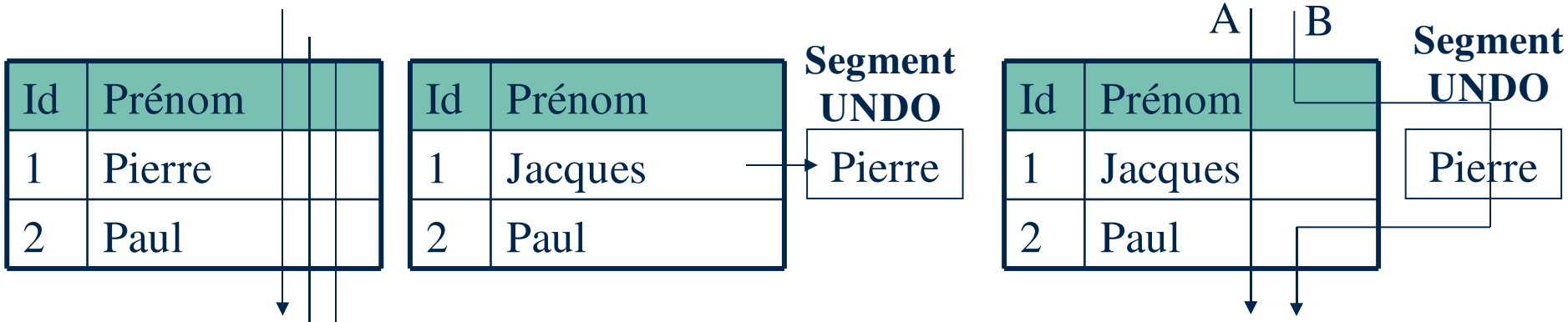
```
set transaction isolation level {0 | 1 | 2 | 3}
```

- Un niveau d'isolation peut aussi être fixé pour l'exécution d'une seule requête :

```
select * from T at isolation {read uncommited |
read commited | repeatable read | serializable}
```

Application au SGBD Oracle

- Oracle supporte les niveaux 1, 2, 3 mais les `read uncommited` (niveau 0) sont impossibles
- Quand un utilisateur modifie des valeurs, tous les autres utilisateurs ont accès aux données non modifiées (utilisation systématique des segments UNDO)



1) Toutes les transations lisent les données de la table

2) La transaction A modifie un enregistrement. L'ancienne valeur est placée dans un segment d'annulation UNDO

3) La transaction A lit la donnée modifiée, la transaction B lit l'ancienne valeur placée dans le segment UNDO

Types de verrous utilisés sous Oracle

- **EXCLUSIVE (X)**

  Empeche toute autre transaction de poser explicitement un verrou sur cette table et d'accéder à cette table en modification

- **SHARE (S)**

  Empeche toute autre transaction de poser un verrou autre qu'un verrou S sur la table et d'y accéder en modification

- **ROW SHARE (RS)**

  Permet l'accès concurrent à la table (modification de lignes différentes dans chacune des transactions) et empeche toute autre transaction de poser sur cette table un verrou exclusif

- **ROW EXCLUSIVE (RX)**

  Permet l'accès concurrent à la table (modification de lignes différentes dans chacune des transactions) et empeche toute autre transaction de poser un verrou sur cette table

- **SHARE ROW EXCLUSIVE (SRX)**

  Empêche la pose d'un verrou autre qu'un verrou d'intention RS

Pose implicite ou explicite (Oracle)

- Pose implicite
  - Les instructions INSERT UPDATE et DELETE posent automatiquement un verrou RX sur la table en cours de modification
  - Une instruction SELECT FROM FOR UPDATE pose un verrou S sur la table
- Pose explicite : l'instruction LOCK

```
LOCK TABLE {table / view} IN {EXCLUSIVE /
  SHARE /ROW SHARE / ROW EXCLUSIVE / SHARE ROW
  EXCLUSIVE} MODE ;
```

Les verrous : problèmes

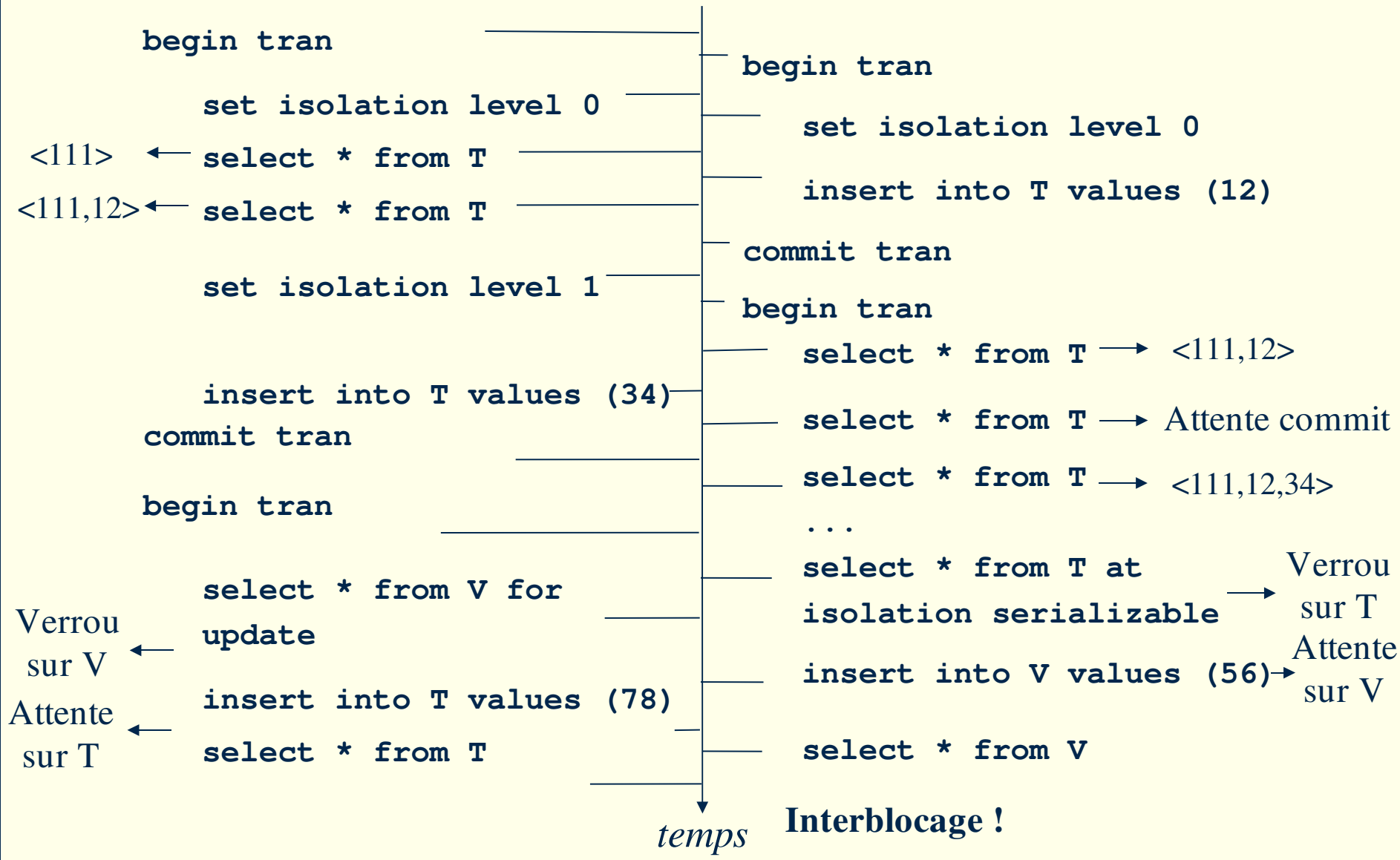
- **Inter-blocage**

T1 détient le granule G1 et attend que T2 libère G2 pendant que T2 attend la libération de G1

- **Famine**

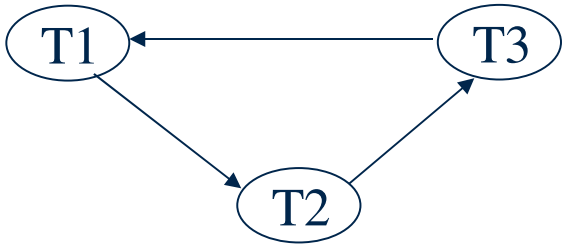
Une transaction est perpétuellement en attente alors que d'autres continuent à s'exécuter

Exemple Sybase



Solutions pour l'inter-blocage

- La *prévention* : imposer à toute transaction de verrouiller en avance tous les éléments dont elle a besoin (n'apposer aucun verrou si un de ces éléments n'est pas libre)
- La *détection* : tester périodiquement si une situation d'inter-blocage s'est produite (solution implémentée dans Sybase et Oracle) :
  - détection de cycles dans un *graphe d'attente*, dont les nœuds représentent les transactions et les arcs la relation *est_en_attente_de*
  - si une telle situation se produit, une des transactions impliquée est avortée



- T1 détient R1 et attend R2
- T2 détient R2 et attend R3
- T3 détient R3 et attend R1

Solutions pour la famine

Cette situation se produit si la priorité est toujours donnée aux mêmes transactions →

- avoir un mécanisme de type "premier arrivé, premier servi" (*First In First Out*)
- adopter une stratégie avec priorité dynamique
- Sybase : après trois tentatives infructueuses de satisfaction d'une demande d'apposition d'un verrou exclusif, toute nouvelle demande de verrou partagé est refusée

2.2 JDBC

JDBC

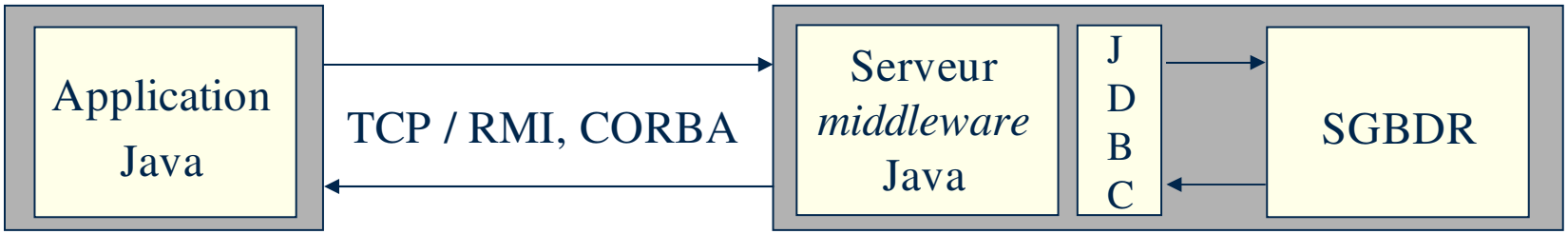
- Objectifs :
  - Fournir un accès homogène aux SGBDR
  - Abstraction des SGBDR cibles
  - Requêtes SQL
  - Simple à mettre en œuvre

  → JDBC (*Java DataBase Connectivity*)

Architectures 2-tier et 3-tier

Architecture 2-tier :

Architecture 3-tier :



Fonctionnement

- Utilisation du *package* `java.sql`
- JDBC interagit avec le SGBDR par un pilote (*driver*)
- Il existe des pilotes pour Oracle, Sybase, Informix, DB2, …
- Selon *JavaSoft*, 4 types de pilotes

Les pilotes JDBC (1/4)

- **Type 1 : *JDBC-ODBC bridge***

Traduisent les appels JDBC en appels ODBC et les transmettent à un driver ODBC

ODBC: *Open DataBase Connection* (accès aux bases de données dans le monde Microsoft)

Les pilotes JDBC (2/4)

- **Type 2 : *native-API partly-Java driver***

Convertissent les appels JDBC en appels spécifiques à la base de données

Nécessitent le chargement de code natif spécifique au SGBD sur la machine où s'exécute l'application cliente

Les pilotes JDBC (3/4)

- **Type 3 : *net-protocol all-Java driver***

Convertissent les appels JDBC en appels à un serveur intermédiaire (*middle-tier*), en utilisant un protocole indépendant du SGBD

Le serveur connecte les clients Java à différentes bases de données et transforme les appels qu'il reçoit pour les rendre conformes au protocole de communication du SGBD considéré

Les pilotes JDBC (4/4)

- **Type 4 : *native-protocol all-Java driver***

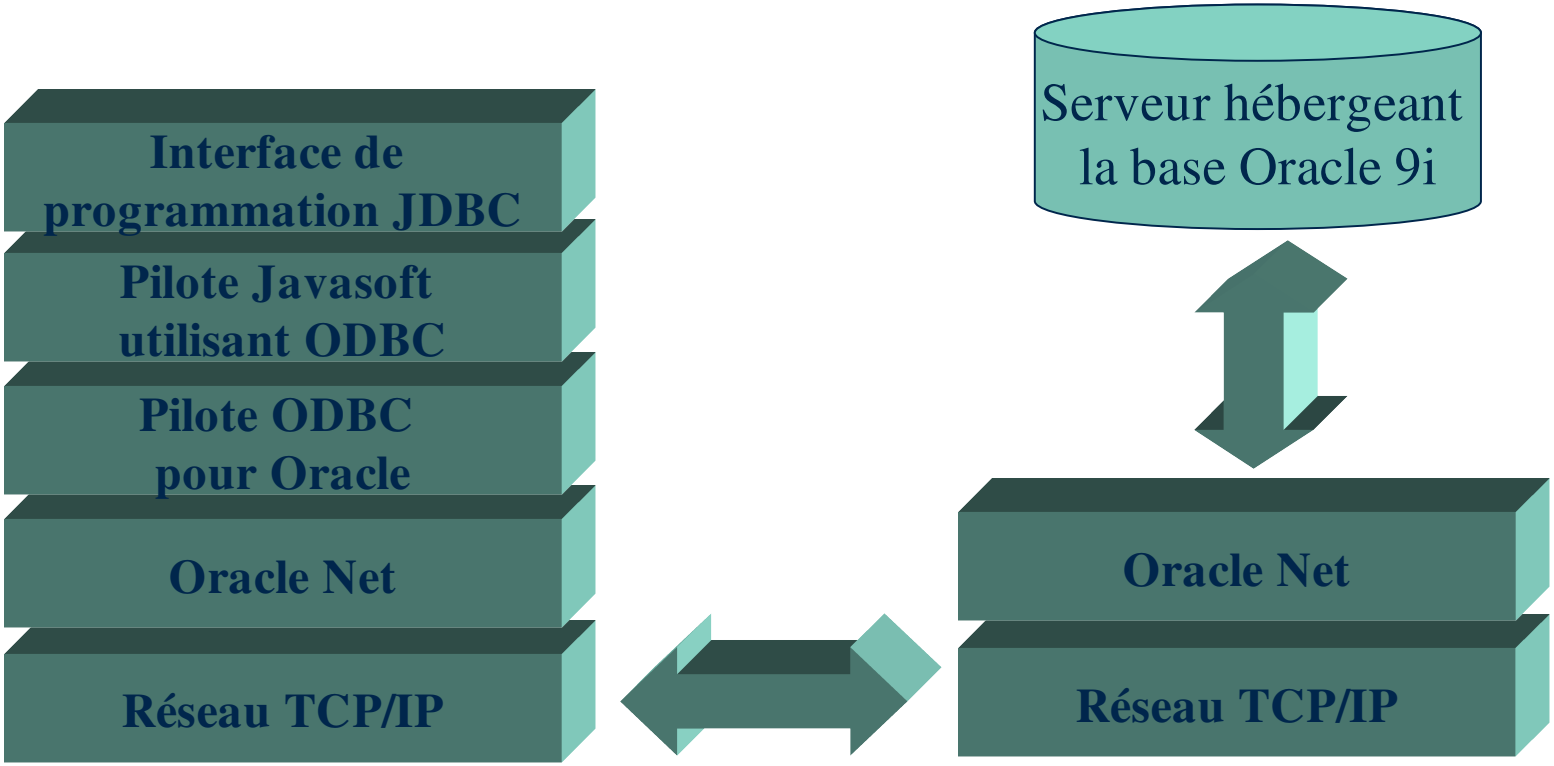
Traduisent directement les appels JDBC en appels au serveur de la base de données, en respectant le protocole de communication du SGBD

Généralement développés par les vendeurs de SGBD

Un pilote de ce type est utilisé dans les exemples

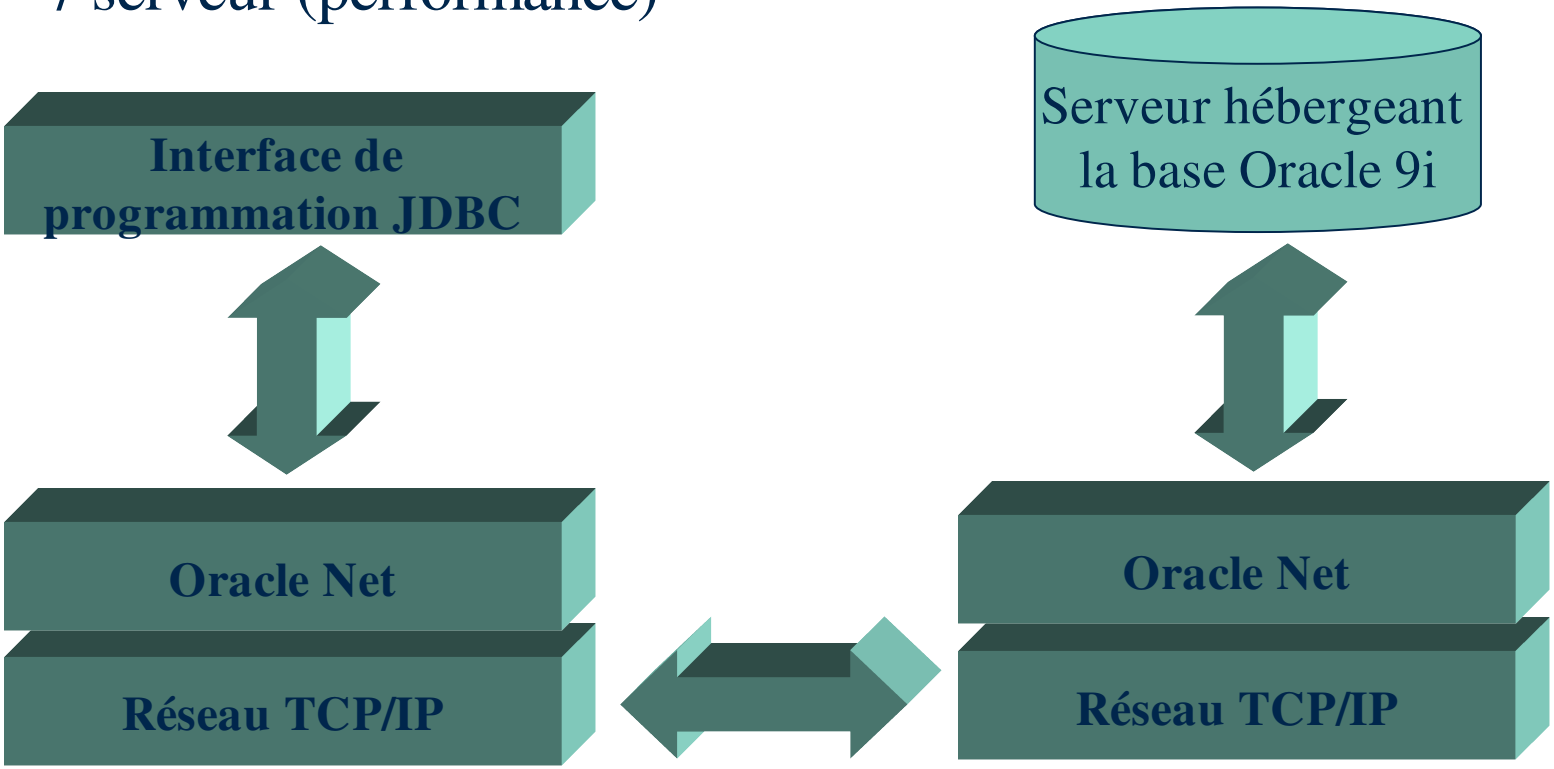
Solutions proposées par Oracle (1/3)

- Pilote Javasoft utilisant ODBC : solution lourde !



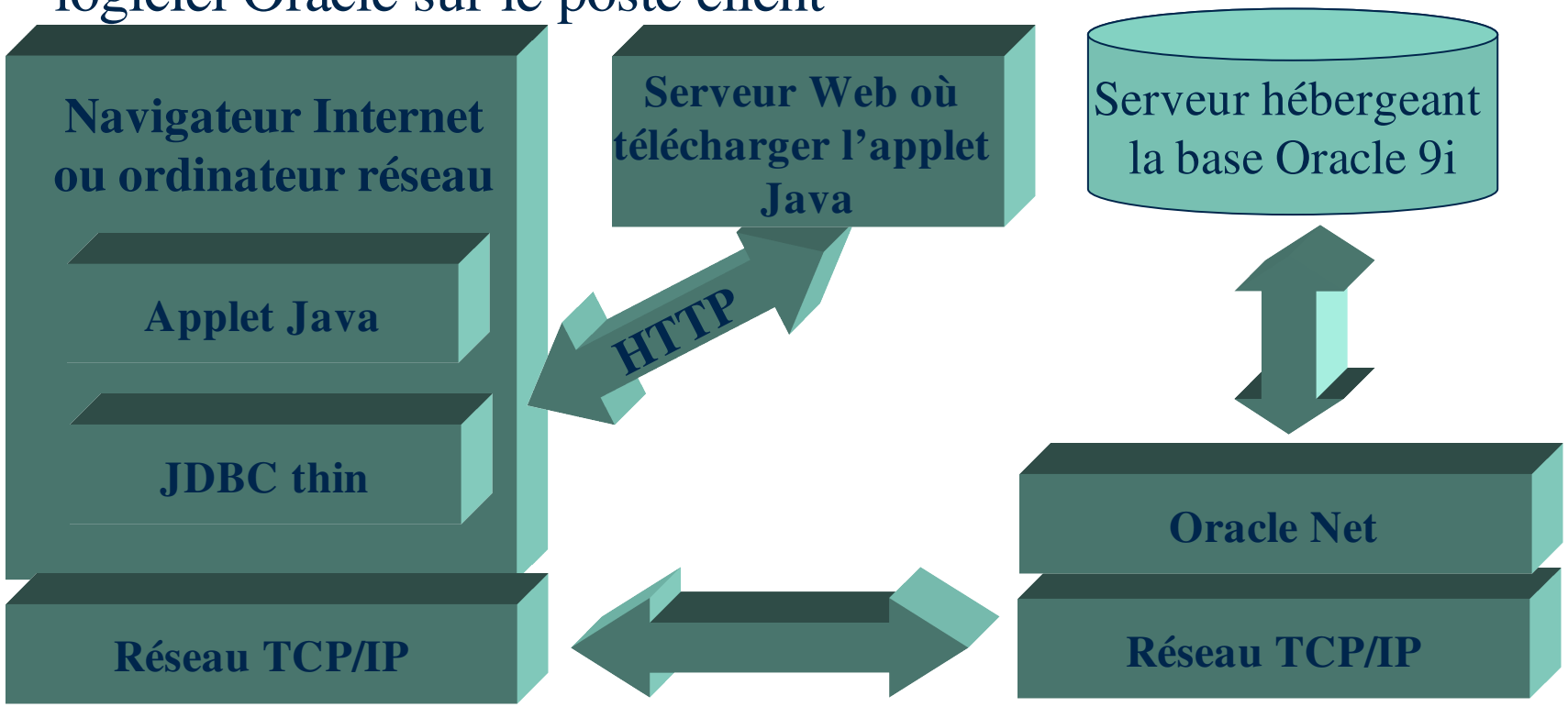
Solutions proposées par Oracle (2/3)

- Pilotes OCI s'appuyant sur Oracle Net : architecture client / serveur (performance)



Solutions proposées par Oracle (3/3)

- Pilote téléchargeable Thin JDBC : évite l'installation d'un logiciel Oracle sur le poste client



Accès aux données

- Charger le *driver*
- Connexion à la base
- Création d'un *statement*
- Exécution de la requête
- Lecture des résultats

Chargement du *driver*

- La classe **`DriverManager`** du package **`java.sql`** gère l'ensemble des pilotes disponibles et se charge d'en sélectionner un qui convienne lors de la connexion à une BD
- Utiliser la méthode **`forName`** de la classe **`Class`** :

```
Class.forName("com.sybase.jdbc.SybDriver");
```

- Lors du chargement d'une classe, les blocs statiques d'initialisation de la classe sont exécutés → c'est par ce moyen que le pilote s'enregistre auprès de la classe **`DriverManager`** en appelant la méthode statique **`DriverManager.registerDriver`**`( )`

Connexion à la base

- Accès via un URL qui specifie :
  - l'utilisation de JDBC
  - le driver ou le type du SGBDR
  - l'identification de la base
- Exemple :

```
String url = "jdbc:sybase:Tds:zeus.scinfo.uhp-
  nancy.fr:5025";
```

- Ouverture de la connexion :

```
Connection con = DriverManager.getConnecion(url,
  user, password);
```

Création d'un *statement*

- 3 types de *statements* :
  - **`statement`** : requêtes simples
  - **`prepared statement`** : requêtes précompilées
  - **`callable statement`** : procédures stockées
- Création d'un *statement* :

```
Statement stmt = con.createStatement();
```

Exécution d'une requête (1/2)

- 3 types d'executions :
  - **`executeQuery`** : pour les requêtes qui retournent un **`ResultSet`**
  - **`executeUpdate`** : pour les requêtes **`INSERT`**, **`UPDATE`**, **`DELETE`**, **`CREATE TABLE`** et **`DROP TABLE`**
  - **`execute`** : pour quelques cas rares (procédures stockées)

Exécution d'une requête (2/2)

- Exécution de la requête :

```
String myQuery = "SELECT prenom, nom, email " +
                 "FROM employe " +
                 "WHERE (nom = 'Dupont') AND
                 (email IS NOT NULL) " +
                 "ORDER BY nom"
ResultSet rs = stmt.executeQuery(myQuery);
```

Lecture des résultats (1/2)

- **`executeQuery`** renvoie un **`ResultSet`** (RS)
- Le RS se parcourt itérativement ligne par ligne
- Les colonnes sont référencées par leur numéro ou par leur nom
- L'accès aux valeurs des colonnes se fait par des méthodes **`getXXX()`** où **`XXX`** représente le type de l'objet

Lecture des resultats (2/2)

```
java.sql.Statement stmt = con.createStatement;

ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM
  Table1");

while (rs.next())

{

   // affiche les valeurs pour la ligne courante

   int i = rs.getInt("a");

   String s = rs.getString("b");

   byte b[] = rs.getBytes("c");

   System.out.println("ROW = "+ i + " " + s + " " +
  b[0]);

}
```

Accès aux méta-données

- La méthode **`getMetaData()`** permet d'obtenir les méta-données d'un **`ResultSet`**
- Elle renvoie des **`ResultSetMetaData`**
- On peut connaître :
  - le nombre de colonnes : **`getColumnCount()`**
  - le nom d'une colonne : **`getColumnName(int col)`**
  - le type d'une colonne : **`getColumnType(int col)`**

Exemple de création de table (1/2)

```
import java.lang.*;
import java.sql.*;
public class CreateTable {
   public static void main (String args[]) {
      String url = "jdbc:sybase:Tds:zeus.scinfo.uhp-
   nancy.fr:5025";
      Connection con;
      Statement stmt;
      String  query = "create table client(nom varchar(20),
   prenom varchar(20), age int)";
      String login="gsimon";
      String password="bddes2";
```

Exemple de création de table (2/2)

```
try {
    Class.forName ("com.sybase.jdbc.SybDriver");}
catch(ClassNotFoundException e) {
    System.err.print("ClassNotFoundException");
    System.err.println(e.getMessage());}
 try {
    con = DriverManager.getConnection (url, login,
password);
    stmt = con.createStatement();
    stmt.executeUpdate(query);}
catch (SQLException ex) {
    System.err.println (" SQLException caught: " +
ex.getMessage());}}
```

Exemple d'insertion de données

```
(…)

stmt = con.createStatement ();

stmt.executeUpdate("insert into client values('Titi',
   'lala', 20)");

stmt.executeUpdate("insert into client values('Dupond',
   'Paul', 30)");

stmt.executeUpdate("insert into client values('Momo',
   'toto', 10)");

(…)
```

Exemple de requête précompilée

```
(…)
PreparedStatement stmt = con.prepareStatement("select *
   from client where nom = ? and prenom = ?");
stmt.setString(1,"Titi");
stmt.setString(2,"Lala");
ResultSet rs= stmt.executeQuery();
while (rs.next()) {
   String n = rs.getString(1); // nom
   String p = rs.getString(2); // prenom
   int a = rs.getInt(3); // age
   System.out.println("nom: "+n+" prenom: "+p+"age: "+a);}
(…)
```

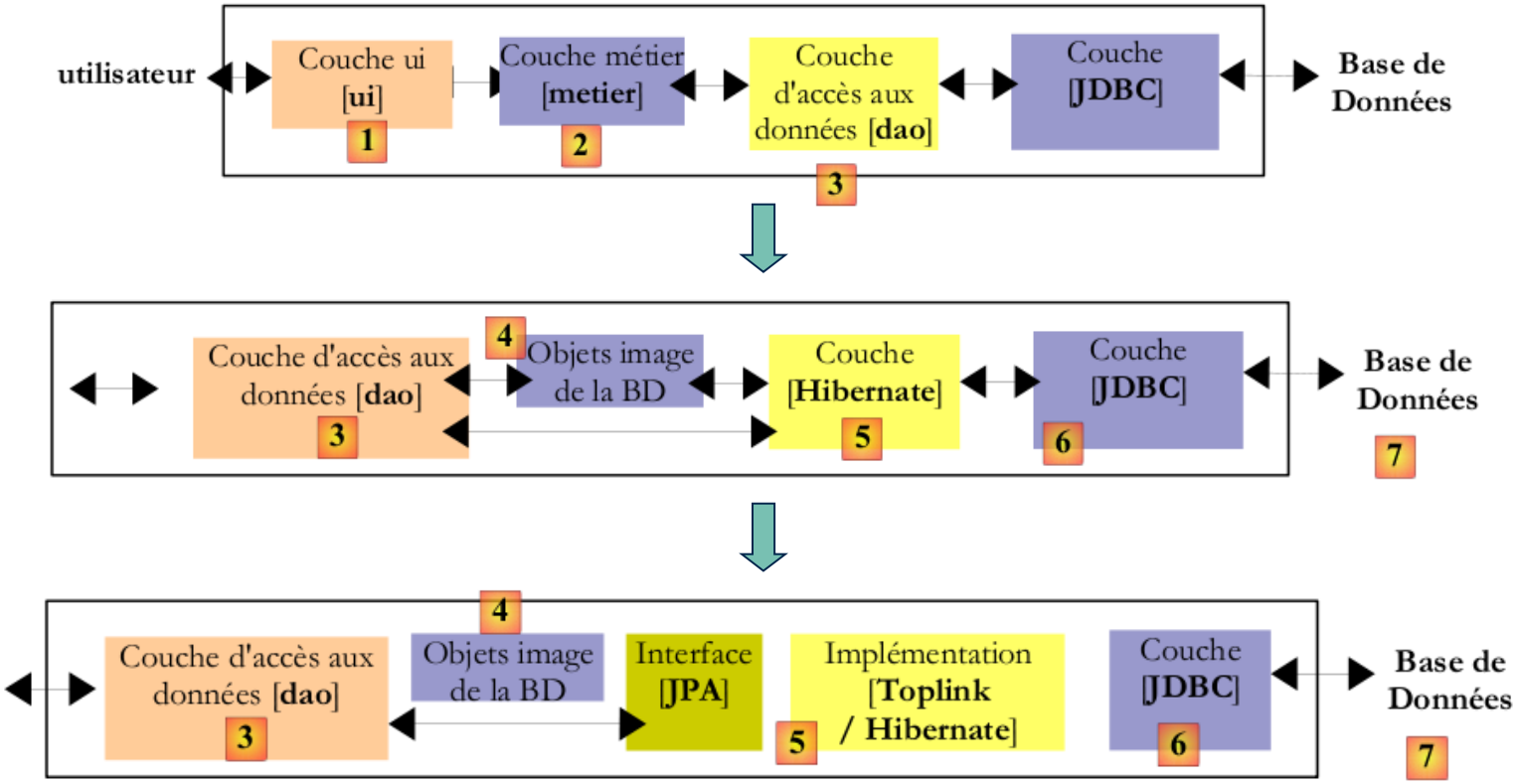
Exemple d'accès aux méta-données

```
(…)
ResultSet rs = stmt.executeQuery("select * from
   authors");
ResultSetMetaData rsmd = rs.getMetaData();
int nbColonnes = rsmd.getColumnCount();
for (int i = 1; i <= nbColonnes; i++) {
   String typeColonne = rsmd.getColumnTypeName(i);
   String nomColonne = rsmd.getColumnName(i);
   System.out.println("la colonne "+i+" de nom
   "+nomColonne+ " est du type "+typeColonne);
}
(…)
```

2.3 Object Relational Mapping

Principe

http://tahe.ftp-developpez.com/fichiers-archive/jpa.pdf



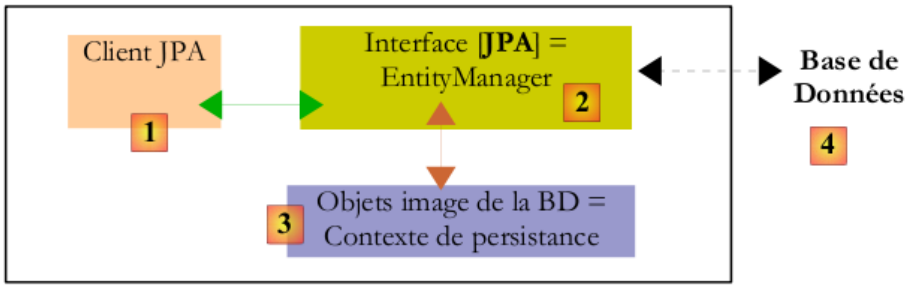
Exemple de définition d'un objet image d'une table

```
1. ...
2.
3. @SuppressWarnings("unused")
4. @Entity
5. @Table(name="Personne")
6. public class Personne implements
    Serializable{
7.
8. @Id
9. @Column(name = "ID", nullable = false)
10. @GeneratedValue(strategy =
    GenerationType.AUTO)
11. private Integer id;
12.
13. @Column(name = "VERSION", nullable = false)
14. @Version
15. private int version;
16.
17. @Column(name = "NOM", length = 30, nullable
    = false, unique = true)
18. private String nom;
19.
20. @Column(name = "PRENOM", length = 30,
    nullable = false)
21. private String prenom;
22.
23. @Column(name = "DATENAISSANCE", nullable =
    false)
24. @Temporal(TemporalType.DATE)
25. private Date datenaissance;
26.
27. @Column(name = "MARIE", nullable = false)
28. private boolean marie;
29.
30. @Column(name = "NBENFANTS", nullable =
    false)
31. private int nbenfants;
32.
33. // constructeurs
34. public Personne() {
35. }
36.
37. public Personne(String nom, String prenom,
    Date datenaissance, boolean marie,
38.         int nbenfants) {
39.     setNom(nom);
40.     setPrenom(prenom);
41.     setDatenaissance(datenaissance);
42.     setMarie(marie);
43.     setNbenfants(nbenfants);
44. }
45.
```

Configuration de la couche JPA (fichier xml)

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence">
3. <persistence-unit name="jpa" transaction-type="RESOURCE_LOCAL">
4.     <!-- provider -->
5.     <provider>org.hibernate.ejb.HibernatePersistence</provider>
6.     <properties>
7.        <!-- Classes persistantes -->
8.        <property name="hibernate.archive.autodetection" value="class, hbm" />
9.        <!-- logs SQL
10.          <property name="hibernate.show_sql" value="true"/>
11.          <property name="hibernate.format_sql" value="true"/>
12.          <property name="use_sql_comments" value="true"/>
13.       -->
14.       <!-- connexion JDBC -->
15.       <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver" />
16.       <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/jpa" />
17.       <property name="hibernate.connection.username" value="jpa" />
18.       <property name="hibernate.connection.password" value="jpa" />
19.       <!-- création automatique du schéma -->
20.       <property name="hibernate.hbm2ddl.auto" value="create" />
21.       <!-- Dialecte -->
22.       <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect" />
23.       <!-- propriétés DataSource c3p0 -->
24.       <property name="hibernate.c3p0.min_size" value="5" />
25.       <property name="hibernate.c3p0.max_size" value="20" />
26.       <property name="hibernate.c3p0.timeout" value="300" />
27.       <property name="hibernate.c3p0.max_statements" value="50" />
28.        <property name="hibernate.c3p0.idle_test_period" value="3000" />
29.    </properties>
30. </persistence-unit>
31. </persistence>
```

Contexte de persistance d'une application



L'interface EntityManager :

void persist(Object entity)	met entity dans le contexte de persistance
void remove(Object entity)	enlève entity du contexte de persistance
<T> T merge(T entity)	fusionne un objet entity du client non géré par le contexte de persistance avec l'objet entity du contexte de persistance ayant la même clé primaire. Le résultat rendu est l'objet entity du contexte de persistance.
<T> T find(Class<T> entityClass, Object primaryKey)	met dans le contexte de persistance, un objet cherché dans la base de données via sa clé primaire. Le type T de l'objet permet à la couche JPA de savoir quelle table requêter. L'objet persistant ainsi créé est rendu au client.
Query createQuery(String queryText)	crée un objet Query à partir d'une requête JPQL (Java Persistence Query Language). Une requête JPQL est analogue à une requête SQL si ce n'est qu'on requête des objets plutôt que des tables.
Query createNativeQuery(String queryText)	méthode analogue à la précédente, si ce n'est que queryText est un ordre SQL et non JPQL.
Query createNamedQuery(String name)	méthode identique à createQuery, si ce n'est que l'ordre JPQL queryText a été externalisé dans un fichier de configuration et associé à un nom. C'est ce nom qui est le paramètre de la méthode.

API de l'interface Query (1/2)

3	int **executeUpdate**() Execute an update or delete statement.
1	List **getResultList**() Execute a SELECT query and return the query results as a List.
2	Object **getSingleResult**() Execute a SELECT query that returns a single result.
	Query **setFirstResult**(int startPosition) Set the position of the first result to retrieve.
	Query **setFlushMode**(FlushModeType flushMode) Set the flush mode type to be used for the query execution.
	Query **setHint**(String hintName, Object value) Set an implementation-specific hint.
	Query **setMaxResults**(int maxResult) Set the maximum number of results to retrieve.
	Query **setParameter**(int position, Calendar value, TemporalType temporalType) Bind an instance of java.util.Calendar to a positional parameter.
	Query **setParameter**(int position, Date value, TemporalType temporalType) Bind an instance of java.util.Date to a positional parameter.
5	Query **setParameter**(int position, Object value) Bind an argument to a positional parameter.
	Query **setParameter**(String name, Calendar value, TemporalType temporalType) Bind an instance of java.util.Calendar to a named parameter.
	Query **setParameter**(String name, Date value, TemporalType temporalType) Bind an instance of java.util.Date to a named parameter.
4	Query **setParameter**(String name, Object value) Bind an argument to a named parameter.

API de l'interface Query (1/2)

1 - getResultList execute un SELECT qui ramène plusieurs objets. Ceux-ci seront obtenus dans un objet List. Cet objet est une interface. Celle-ci offre un objet Iterator qui permet de parcourir les éléments de la liste L sous la forme suivante :

```
1.          Iterator iterator = L.iterator();
2.          while (iterator.hasNext()) {
3.              // exploiter l'objet iterator.next() qui représente l'élément courant de la liste
4. ...
5. }
```

28/305

La liste L peut être également exploitée avec un for :

```
1.     for (Object o : L) {
2.         // exploiter objet o
3. }
```

2 - getSingleResult exécute un ordre JPQL / SQL SELECT qui ramène un unique objet.

3 - executeUpdate exécute un ordre SQL update ou delete et rend le nombre de lignes affectées l'opération.

4 - setParameter(String, Object) permet de donner une valeur à un paramètre nommé d'un ordre JPQL paramétré

5 - setParameter(int, Object) ; le paramètre n'est pas désigné par son nom mais par sa position dans l'ordre JPQL.

Exemple de code client

```
1. package tests;
2.
3. import java.text.ParseException;
4. import java.text.SimpleDateFormat;
5.
6. import javax.persistence.EntityManager;
7. import javax.persistence.EntityManagerFactory;
8. import javax.persistence.EntityTransaction;
9. import javax.persistence.Persistence;
10.
11. import entites.Personne;
12.
13. public class InitDB {
14. // constantes
15. private final static String TABLE_NAME =
      "jpa01_personne";
16.
17. public static void main(String[] args) throws
      ParseException {
18.     // Unité de persistance
19.     EntityManagerFactory emf =
      Persistence.createEntityManagerFactory("jpa");
20.     // récupérer un EntityManagerFactory à partir de
      l'unité de persistance
21.     EntityManager em = emf.createEntityManager();
22.     // début transaction
23.     EntityTransaction tx = em.getTransaction();
24.     tx.begin();
25.     // supprimer les éléments de la table des personnes
26. em.createNativeQuery("delete from " +
      TABLE_NAME).executeUpdate();
27.     // créer deux personnes
28.     Personne p1 = new Personne("Martin", "Paul", new
    SimpleDateFormat("dd/MM/yy").parse("31/01/2000"), true,
      2);
29.     Personne p2 = new Personne("Durant", "Sylvie", new
    SimpleDateFormat("dd/MM/yy").parse("05/07/2001"),
      false, 0);
30.     // persistance des personnes
31.     em.persist(p1);
32.     em.persist(p2);
33.     // affichage personnes
34.     System.out.println("[personnes]");
35.     for (Object p : em.createQuery("select p from
      Personne p order by p.nom asc").getResultList())
    {
36.         System.out.println(p);
37.     }
38.     // fin transaction
39.     tx.commit();
40.     // fin EntityManager
41.     em.close();
42.     // fin EntityManagerFactory
43.     emf.close();
44.     // log
45.     System.out.println("terminé ...");
46. }
47. }
```

2.4 PHP

PHP

- PHP est un langage interprété, exécuté du côté serveur
- Principaux atouts :
  - gratuit et *open source*
  - simplicité d'écriture des scripts
  - possibilité d'inclure le script PHP au sein d'une page html
  - simplicité d'interfaçage avec des bases de données (MySql, Oracle, Sybase, dBase, Informix etc.)
  - Intégration au sein de nombreux serveurs *web* (Apache, Microsoft IIS ...)

Interprétation du code par le serveur

- Instructions incluses dans un code HTML à l ’aide de balises spéciales (<?php et ?>)
- Le fichier HTML+PHP doit avoir l’extension .php
- Principe :
  - le serveur lit le fichier PHP
  - des qu’il rencontre une balise PHP, il passe en mode PHP
  - chaque instruction est transmise à l’interpréteur qui l'exécute
  - le serveur transmet le résultat au client (dans le fichier HTML)
- **Le script PHP n’est jamais visible par l’utilisateur !**

Un exemple simple

```
<html>
<head><title>Exemple</title>/</head>
<body>
<?php
echo "Hello world";
?>
</body>
</html>
```

; à la fin de chaque instruction PHP

Variables et tableaux

- Les variables commencent toujours par $
- Elles ne sont pas déclarées !
- Les tableaux :

```
$tab[0][0] = 12;

$tab[0][1] = "php"

$tab[1][0] = 1234.56

$tab[1][0] = "au revoir"
```

Opérateurs

- Opérateurs de calcul : `+ - * / =`
- Opérateurs d'assignation : `+= -= *= /= %= |= ^= &= .=`
- Opérateurs d'incrémentation : `++ --`
- Opérateurs de comparaison : `== <= > >= !=`
- Opérateurs logiques : `||` (OR) `&&` (AND) XOR `!`

Structures conditionnelles (1/4)

- L ’instruction if ... elseif ... else

```
if (condition realisee) {
   liste d ’instructions
}
elseif (autre condition realisee) {
   autre serie d ’instructions
}
...
else (derniere condition realisee) {
   derniere serie d ’instructions
}
```

Structures conditionnelles (2/4)

- L'instruction switch

```
switch (variable) {
   case valeur1:
   liste d 'instructions
   break;
   case valeur2:
   liste d 'instructions
   break;
...
   default:
   liste d 'instructions
   break;
}
```

Structures conditionnelles (3/4)

- La boucle for

```
for (compteur; condition; modification du
   compteur) {
   liste d 'instructions
}
```

Ex :

```
for ($i=1; $i<6; $i++){
   echo "$i<br>";
}
```

Structures conditionnelles (4/4)

- L ’instruction while

```
while (condition realisee) {
   liste d ’instructions
}
```

Les fonctions (1/1)

- Declaration :

```
function nom_de_la_fonction(type1 argument1,
   type2 argument2, ...) {

   liste d 'instructions

}
```

- Renvoi d'une valeur :

```
return valeur;
```

- Appel :

```
[$retour = ] nom_de_la_fonction(argument1,
   argument2, ...)
```

Les fonctions (2/2)

- Passage de paramètres par référence : faire précéder la variable du caractère &
- Exemple :

```
<?
function dire_texte($qui, &$texte) {
   $texte = "Bienvenue $qui";
}
$chaine = "Bonjour ";
dire_texte("cher phpeur", $chaine);
echo $chaine; // affiche "Bienvenue cher phpeur"
?>
```

Interfaçage avec un SGBD (ex : mysql)

- Les principales étapes :
  - se connecter au serveur de la base de données : *mysql_connect*
  - choisir la base : *mysql_select_db*
  - exprimer une requête : *mysql_query*
  - récupérer les résultats (pour une lecture) : *mysql_fetch_row*
  - se déconnecter : *mysql_close*

Un exemple complet (1/2)

```
<html>
   <head>
   <title>Liens</title>
   <head>
<body>

<table border="1" cellpadding="0"
 cellspacing="0">
<tr><th>Nom du site</th>
<th>URL</th></tr>

<?php

// Déclaration des paramètres de
 connexion

$host = la_machine;

// Généralement la machine est
   localhost
// c'est-a-dire la machine sur
   laquelle
 le script est hébergé
$user = votre_login;
$bdd = Nom_de_la_base_de_donnees;
$password  = Mot_de_passe;

// Connexion au serveur

mysql_connect($host, $user,$password)
   or
 die("erreur de connexion au
   serveur");
mysql_select_db($bdd) or die("erreur
   de
 connexion a la base de donnees");

// Creation et envoi de la requete

$query = "SELECT nom,url FROM sites
 ORDER BY nom";
$result = mysql_query($query);
```

Un exemple complet (2/2)

```
// Recuperation des resultats

while($row = mysql_fetch_row($result)){
$Nom = $row[0];
$Url = $row[1];
echo "<tr>\n
<td><a href=\"$Url\">$Nom</a></td>\n
<td>$Url</td>\n
</tr>\n";
}

// Deconnexion de la base de donnees

mysql_close();
?>

</tr>
</table>
</body>
</html>
```

Exemple de réception des variables d'un formulaire

Création et envoi du formulaire

```
<html>
<head>
<title>Formulaire</title>
</head>
<body>
<p>Bonjour, quel est ton nom ?
</p>
<form method="post"
action= "exemple.php">
<input type="text"
name="mavariable" value="Toto">
<input type="submit" value="OK">
</form>
</body>
</html>
```

Reception du contenu (fichier `exemple.php`)

```
<html>
<head>
<title>Réception formulaire</title>
</head>
<body>
<?php
echo "Bonjour ".$mavariable;
?>
</body>
</html>
```