UNIVERSITÉ DE LORRAINE

FST, MIAE, Département Informatique

Master Informatique, Spécialité LMFI

Support de cours de l'EC Bases de données avancées

Nacer.Boudjlida@loria.fr

Contenu :

1. Introduction aux SGBD relationnels (Rappel)
2. Bases de données réparties
3. Annexes :
    a. Traitement des requêtes dans les SGBD centralisés
    b. Gestion des transactions dans les SGBD centralisés

SUPPORT DELIBEREMENT INCOMPLET. LES COMPLEMENTS SONT DONNES EN COURS

UNIVERSITÉ
DE LORRAINE

Introduction aux SGBDR

Nacer Boudjlida
http://www.loria.fr/~nacer

Université Henri Poincaré Nancy 1
FST, UFR STMIA

Introduction aux SGBDR

1. Rôle/Fonctions d'un SGBD
2. Typologie des SGBD
3. Situation dans l'architecture des applications
4. Langages Relationnels
5. Architecture d'un SGBD
6. "'Métiers du domaine"' et compétences

1- Rôle/Fonctions d'un SGBD

1. Décrire des "objets", leurs relations, des contraintes, etc.
2. Manipuler les données
3. Confidentialité
4. Intégrité
5. Accès concurrents et sécurité de fonctionnement

2- Typologie des SGBD

- Modèle de représentation et de manipulation de données ⟶ Classe de SGBD
  1. Hiérarchique, Réseau ⟶ CODASYL
  2. Relationnel ⟶ SGBD Relationnel
  3. A objets ⟶ SGBDOO
  4. [Relationnel "étendu" ⟶ "Objet-Relationnel]
  5. Logique ⟶ SGBD Déductif
  6. Non ou semi-tructurées ⟶ SGBD pour XML (?)

3- Situation dans l'architecture des applications

4- Langages Relationnels

Type	**Fondement**
Algébrique	Théorie des ensembles
Prédicatifs a) à variable n-uplet b) à variable domaine	Logique du 1er ordre

- SQL ≃ "dialecte" fondé sur l'algèbre et le calcul prédicatif à variable n-uplet

4.1- Algèbre relationnelle

- Caractéristiques :
  - Opérande(s) : Relation(s)
  - Résultat : Relation
  - Opérateur : Opérateur du calcul relationnel
- Ensemble minimal d'opérateurs :
  1. Restriction/sélection (σ)
  2. Projection (Π)
  3. Produit cartésien (X)
  4. Union ($\cup$)
  5. Différence ($\setminus$)
- Remarque : Jointure ($\bowtie = \sigma(X(\ldots))$

4.2- Syntaxe d'un langage algébrique

- Si R est un nom de relation alors R est une expression algébrique (ea)
-
-
-

4.3- De l'algèbre à SQL

- R $\longrightarrow$
- $\sigma_C(R)$ $\longrightarrow$
- $\Pi_L(R)$ $\longrightarrow$
- $R1\,X\,R2)$ $\longrightarrow$
- $R1 \setminus R2$ $\longrightarrow$
- $R1 \cup R2$ $\longrightarrow$
- En réalité, dans un SGBD : de SQL vers l'algèbre!

5- Architecture d'un SGBD

6. Les "'métiers du domaine"' et compétences

1. Application
   -
   -
2. Système
   - –
     –
   -

"Métiers du domaine" et Compétences

3. Environnements de développement : Conception/développement d'outils
   -
   -
   -

Bases de données réparties

Nacer.Boudjlida@loria.fr

http://www.loria.fr/~nacer

Université de Lorraine, FST/MIAE, ESIAL

Support de cours. Des compléments importants seront donnés en cours.

Chapitre I : Introduction

Bases de données distribuées : Définitions

- Intégration des technologies bases de données (BDD) et réseau
- Intégration des données d'une entreprise sans centralisation
- Distribution "naturelle" :
  - "Agence mère" et ses filiales
  - Unité centrale et sous-système d'entrées/sorties, etc.
- *Définition orientée bases de données* :

  Système de traitement distribué = *(1) Unités d'exécution* de programmes, *(2) autonomes*, éventuellement *(3) hétérogènes*, *(4) reliés par un réseau* de communication et *(4) coopérant* à la réalisation de tâches.

Bases de données distribuées : Définitions (fin)

- *Base de données distribuée (DDB)* :

  Collection de BDD logiquement *reliées* et physiquement *distribuées* sur un réseau
- *SGBD Distribué* :

  Système logiciel de gestion rendant les applications insensibles à la distribution des données (TRANSPARENCE)
- *Base de données distribuée N'EST PAS* :
  - {Fichiers indépendants} sur des sites différents,
  - Base centralisée accessible via le réseau

QUOI distribuer ?

- Données
- Traitements
- Fonctions du système
- Contrôle et coordination des tâches

Problèmes liés à la distribution

- *Conception des bases distribuées* :
  - Partitionnement des données (FRAGMENTATION, LOCALISATION)
  - DUPLICATION totale/partielle
  - Administrateur global, local
- *Dictionnaire(s)*
  - Extension avec des informations sur les sites, la fragmentation, la duplication et la localisation des données
  - Centralisé ou distribué ?
  - Copie simple ou multiple ?

Problèmes liés à la distribution (suite)

- *Traitement des requêtes* :
  - *Objectif* : Optimiser en exploitant le parallélisme
  - Problème “NP-complexe” : Approches heuristiques
  - *Facteurs* :
    - Localisation/Duplication des données
    - Coût des communications
    - Disponibilité locale d’informations suffisantes
  - *Stratégies* : Sous-requêtes, jointures parallèles, semi-jointure, etc.

Problèmes liés à la distribution (suite)

- *Accès concurrents*
  - Contrôle réparti des accès concurrents prenant en compte la duplication
  - Techniques : verrouillage, estampillage
  - Traitement du deadlock distribué ?

Problèmes liés à la distribution (suite)

- *Sécurité et reprise en cas d’incident*
  - Nouveaux types d’incidents : perte de messages, panne d’une liaison, etc.
  - Préservation des bases des sites opérationnels en cas de panne d’un site
- *Confidentialité* sur un réseau

Problèmes liés à la distribution (fin)

- *Hétérogénéité*
  - Matériels/Logiciels
  - Modèles de données et Langages de manipulation
  - Introduite aussi dans les systèmes MULTI-BASEs : regroupement de BDD centralisées autonomes
  - Mécanimes de traduction (données, programmes)
- *Systèmes hôtes* :

  Quelles solutions (homogènes) pour supporter les applications distribuées et celle non distribuées (TRANSPARENCE) ?

Bases de données réparties : Contenu

Chapitre II : Typologie des SGBD distribués

Caractéristiques d'un SGBD distribué

1. *Transparence du réseau*
   - Nommage des objets
   - Localisation des objets
   - Services (Exemple : *cp f1,f2; rcp site:f1, site:f2*)

2. *Transparence de la duplication* : Points de vue
   - Utilisateur : *"Duplication ? Connais pas !"*
   - Système : implications en recherche, en mise à jour, en gestion des accès concurrents

Caractéristiques d'un SGBD distribué

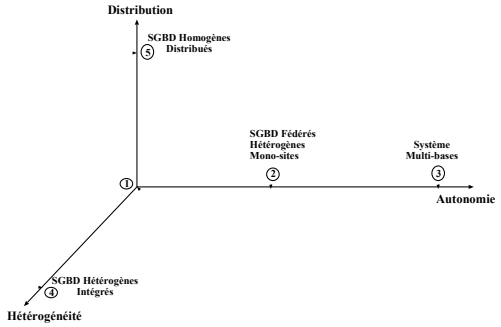
3. *Transparence de la fragmentation* dans le traitement des requêtes
   - Requête sur UNE relation (utilisateur)
   - Décomposée en sous-requêtes (système)
   - Sous-requêtes évaluées sur les fragments (système)
   - Union des réponses (système)

4. *Transparence des langages dans le cas de DDB hétérogènes*

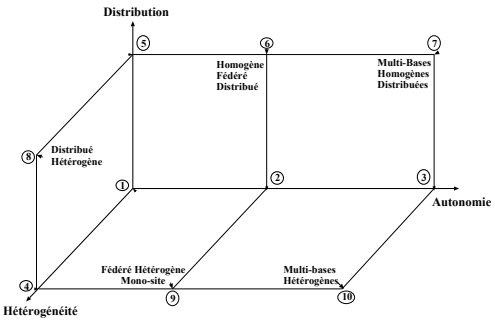
Critères de classification

1. *AUTONOMIE*
   - Concerne la distribution du contrôle
   - Mesure le degré d'indépendance
   - Fonction de :
     - Volume des échanges inter-sites
     - Capacités d'exécution indépendamment des autres sites
2. *DISTRIBUTION* : (ou pas) des données
3. *HETEROGENEITE*
   - Hardware, protocoles réseaux
   - Modèles de données : puissances d'expression
   - Langages : paradigmes, dialecte(s)

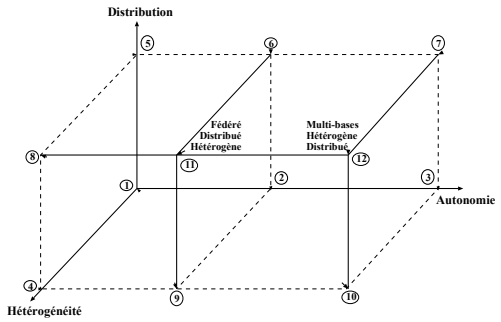
Typologie (1/3)



Typologie (2/3)

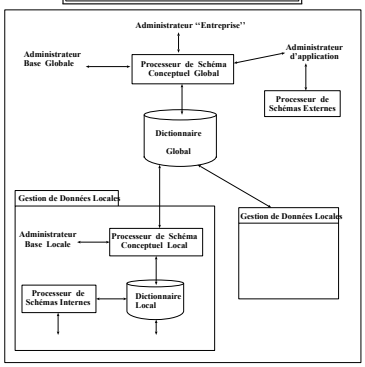


Typologie (3/3)



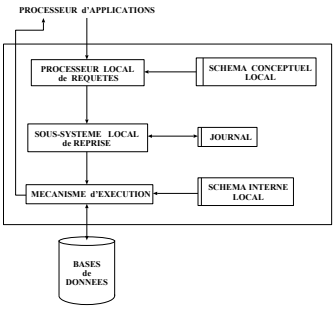
Architecture de référence ANSI/SPARC (1/2)

Architecture de référence (2/2)



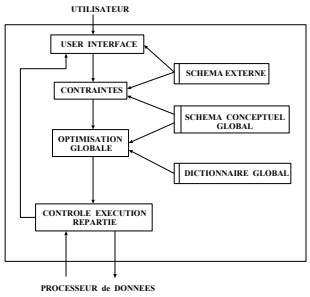
Composants fonctionnels d'un SGBD distribué

1. *Processeur de données*



Composants fonctionnels d'un SGBD distribué

2. *Processeur d'applications*



Chapitre III : Distribution de données

III.1- Fragmentation et duplication de données

- *DUPLICATION TOTALE* :
  - **(+)** Disponibilité des données
  - **(+)** Parallélisme en lecture
  - **(+)** Moindre flux sur le réseau
  - **(-)** Coût des mises à jour

- *FRAGMENTATION* :
  - Découpage d'une relation R en *fragments* $R_1, R_2, \cdots, R_n$
  - *Fragmentation Horizontale* : Reconstruction par *UNION*
  - *Fragmentation Verticale* : Reconstruction par *JOINTURE*

Fragmentation horizontale

- Peut être définie par une sélection
- Fragments disjoints ou non (*Duplication partielle*)

Fragmentation verticale

- Forme la plus simple : Décomposition de R
- "Identifiant" (clé) dans chaque fragment

Fragmentation mixte et duplication

1. *FRAGMENTATION MIXTE* : horizontale et verticale
2. *FRAGMENTATION et DUPLICATION* :
   - Duplication de fragment(s)
   - Fragmentation de duplicata, etc.

- Convention de *nommage des fragments et des copies* :
  - *Désignation_Objet*•F_i : Fragment
  - *Désignation_Objet*•C_i : Copie
  - Exemple : *Site2.Produit.F3.C4*

III.2- Transparence et autonomie

1. *Transparence* :
   - du réseau
   - de la localisation
   - connaissance minimale sur les lieux de rangement des données
2. *Autonomie* :
   - Liée à la transparence
   - Degré d'indépendance vis-à-vis du système distribué

- *Analyse selon les points de vue* :
  (1) Nommage des objets
  (2) Fragmentation et Duplication des données
  (3) Localisation

(1/3) Transparence et autonomie : Nommage des objets

- Assurer *l'unicité* des noms
- *Solution 1* : Serveur de noms centralisé
  (-) Autonomie locale
  (-) Surcharge du serveur ; Blocage en cas de panne

- *Solution 2* : *Nom_du_site*•*Nom_Objet*
  (+) Pas de contrôle centralisé : Meilleure autonomie locale
  (-) Pas de Transparence du réseau

(2/3) Transparence : Fragmentation et duplication

- *DUPLICATION* :
  - Utilisateur : Pas de désignation de copie
  - *Quelle copie accéder ?*
  - *Quelle(s) copie(s) mettre à jour ?*

- *FRAGMENTATION* :
  - Utilisateur : Pas de désignation de fragment
  - *Comment localiser ?*

(3/3) Transparence : Localisation

- Définition d'*alias* (sans nom du site)

  (+) (Légère) transparence à l'utilisation

  (+) Tranparence si changement de localisation

- *Exemple* : Site1 : alias *compte_local* pour le fragment Site1.compte.F1
- *Principe de la localisation*
  1. Remplacer *compte_local* par *Site1.compte.F1*
  2. Si *Site1.compte.F1* dupliqué
     - Accès à la table des copies (dictionnaire)
     - Choisir une copie
  3. Si copie fragmentée $\Longrightarrow$ Consulter la table des fragments

Schéma d'algorithme de localisation

Chapitre IV : Traitement des requêtes

- Processus de traitement des requêtes :

Traitement des requêtes

- *PLAN d'EXECUTION REPARTI* :
  - {*Traitements Locaux* et Opérations de *Communication de données* intermédiaires}
  - Comprend le *nécessaire* pour les *exécutions locales* et la *synchronisation globale*
- *Et la fragmentation* ?
  - *Représentation canonique* de requête algébrique
  - Identification et localisation des fragments
  - *Extension* de la représentation canonique par les sous-arbres de *reconstruction des fragments*

Traitement des requêtes : Exemple

- $Prod1 = \Pi_{prod\#,libelle}(Produit(\underline{prod\#}, libelle, pu))$
- $Prod2 = \Pi_{prod\#,pu}(Produit)$
- $Stock1 = \sigma_{dep\#=4}(Stock(\underline{prod\#, dep\#}, qte))$
- $Stock2 = \sigma_{dep\#<>4}(Stock)$
- *Libellés des produits du depôt 4 ?*

```
select libelle from Produit p, Stock s
where s.prod# = p.prod# and s.dep# = 4
```

$\Pi_{libelle}(\bowtie_{(prod\#=prod\#)} (Produit, \sigma_{(dep\#=4)}(Stock)))$

Traitement des requêtes : Règles de transformation

- *REGLES "CLASSIQUES"* de transformation
- *AUTRES REGLES* :
  1. Condition de sélection et expression de fragmentation *identiques* $\Longrightarrow$ *Sélection INUTILE*
  2. Fragmentation horizontale : *Résultat vide si*
     - Sélection
     - Condition de sélection et expression de fragmentation *CONTRADICTOIRES*
  3. Fragmentation verticale : *Résultat vide si*
     - Fragmentation verticale de liste L'
     - Projection de liste L
     - $L \setminus \{Attribut(s)\ de\ reconstruction\}$ $\not\subset$ $L' \setminus \{Attribut(s)\ de\ reconstruction\}$

Optimisation : Objectifs et facteurs

- Minimiser une fonction coût
- Fonction générale : $\sum_{i=1}^{n} Temps\_Execution_i$ (n : nombre de sites impliqués)
- Autres : Tenir compte
  - du parallélisme
  - des coûts des transferts
  - des *profils* des fragments (Taille, nombre de n-uplets, etc.)
  - de la taille des résultats intermédiaires
  - de l'instant de l'optimisation (compilation/exécution)
  - de la topologie du réseau
  - du coût de l'optimisation, etc.

Optimisation : Types de décisions

- *Décisions incluses dans le plan d'exécution*
  - Ordre des jointures
  - Stratégie de jointure (voir plus loin)
  - Sélection des copies (site le plus proche, le moins engorgé)
  - Choix des sites d'exécution (coûts des communications)
  - Choix des algorithmes d'accès répartis
  - Choix du mode de transfert (tuple/tuple, paquets)

Stratégies d'évaluation des requêtes

- En centralisé : coût des accès
- Cadre distribué :
  - Transmission de données
  - Traitement parallèle des (sous-)requêtes
- Attention particulière aux jointures
- *STRATEGIES* :
  1. Jointures simples avec transferts
  2. Jointures parallèles
  3. Semi-Jointures

Stratégie 1/3 : Jointures simples

- Pas de fragmentation ni de duplication
- SITE S_Q : *Produit* $\bowtie$ *(Depot* $\bowtie$ *Stock)*
- *Produit* : Site S_{Pr}
- *Depot* : Site S_{De}
- *Stock* : Site S_{St}
- *Site émetteur de la requête* : S_Q

Stratégie 1/3 : Jointures simples

- Jointures simples : Stratégie 1
  1. Transfert des relations sur S_Q
  2. Optimisation sur S_Q
  3. Exécution locale

Stratégie 1/3 : Jointures simples

- Jointures simples : Stratégie 2
  1. Transfert de *Stock* vers S_{De}
  2. Calcul de *TEMP1 = (Depot* $\bowtie$ *Stock)*
  3. Transfert de *TEMP1* vers S_{Pr}
  4. Calcul de *TEMP2 = (Produit* $\bowtie$ *TEMP1)*
  5. Transfert de *TEMP2* vers S_Q

Jointures simples : Discussion

- *STRATEGIE 1* :
  - Reconstruction éventuelle des index sur S_Q
  - Volume des transferts
- *STRATEGIE 2* : Volume des transferts

Stratégie 2/3 : Jointures parallèles

- $(R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4)$
- R_i sur site S_i
- Jointures parallèles :
  1. Sur S_1 : $(R_1 \bowtie R_2)$
  2. Sur S_3 : $(R_3 \bowtie R_4)$
  3. Envoi des nuplets vers S_Q au fur et à mesure du calcul
  4. S_Q évalue $(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)$ *parallèlement* aux évaluations sur S_1 et S_3

Stratégie 3/3 : Semi-Jointure ($\propto$)

- $(R(X) \propto S(Y)) = \Pi_X(R \bowtie S)$
- Stratégie permettant d'éviter le transfert de tuples non utiles pour la jointure
- *EXEMPLE* :
  - $R_1 \bowtie R_2$
  - R_1, R_2 sur sites S_1, S_2, respectivement
  - Stratégie Semi-Jointure, résultat sur S_1
    1. S_1 : $TEMP1 = \Pi_{X \cap Y}(R_1)$
    2. S_1 : Envoi de *TEMP1* à S_2
    3. S_2 : $TEMP2 = R_2 \propto TEMP1$
    4. S_2 : Envoi de *TEMP2* à S_1
    5. S_1 : $RES = R_1 \bowtie TEMP2$

Stratégie 3/3 : Semi-Jointure ($\propto$)

- Exemple :
- Correction de la stratégie : Démontrer que $R_1 \bowtie R_2 = (R_1 \bowtie (R_2 \propto (\Pi_{X \cap Y}(R_1))))$?

Stratégie 3/3 : Discussion

- Hypothèse : Sélectivité faible
- $CARD(R_1 \bowtie R_2) < CARD(R_2)$
- Transfert de $(R_2 \propto R_1)$ moins coûteux que celui de R_2
- Ce gain devrait compenser le coût du transfert de $\Pi_{X \cap Y}(R_1)$
- Augmentation du nombre d'opérations
- MAIS sur de plus petites relations
- Gain proportionnel à la sélectivité de la jointure

Traitement des requêtes : Cas des Vues

- *Rappel* : Vue = relation calculée
- Vue définissable sur fragments répartis
- Traitement de requêtes sur vues
  1. Par *ré-écriture*
  2. En utilisant des *vues concrètes*
  3. En utilisant des *clichés* (*Snapshots*)

1/3. Requêtes sur vues et ré-écriture

1. Requête Q_V sur vue ré-écrite en une requête Q_R sur relations
2. Traiter Q_R comme une requête répartie

2/3. Requêtes sur vues et vues concrètes

- Calcul et stockage des valeurs des vues
- Mises à jour systématiques
- Duplication éventuelle sur les sites utilisateurs fréquents
- *SOLUTION ACCEPTABLE* si vue définie sur beaucoup de fragments (Diminution du coût de son "calcul")

3/3. Requêtes sur vues et clichés

- *Cliché* : Vue mise à jour périodiquement
- Réduction du coût par *mises à jour différentielles* (Vue sur une relation, définie par $\Pi_L(\sigma_E(\cdots))$
  - Estampillage des tuples de la relation
  - Estampillage du cliché
  - Calcul d'une $\triangle$relation contenant les tuples du cliché à modifier
  - Mise à jour du cliché (et de ses copies)

3/3. Requêtes sur vues et clichés

Produit

estampille	#prod	libellé	pu
10:00	11	P11	10
10:15	22	P22	20
10:30	33	P33	30
11:00	44	P44	40

Cliché 10:30

#prod	pu
11	10
22	20
33	30

$\triangle$ **Produit 11:00**

#prod	pu
44	40

Cliché 11:00

#prod	pu
11	10
22	20
33	30
44	40

Chapitre V : Contraintes d'intégrité et Confidentialité

Confidentialité

1. *Identification des utilisateurs distants* :

   Sous-transactions d'une transaction T_{Si} comportent l'identification de l'auteur de T_{Si} (Nom, Mot de passe)

2. *Identification du site émetteur* :

   Un site S_i n'accepte des (sous-)transactions de S_j que si S_i connaît S_j

Contraintes d'intégrité

- CI = Formule du calcul relationnel
- Vérifier CI :

  Produit cartésien des relations impliquées satisfait la formule

- *Solution immédiate* : CI satisfaite si

  $Vide(\sigma_{\neg CI}$(X (Relations impliquées)))

- $\sigma_{\neg CI}$(X (Relations impliquées)) : comme une requête répartie
- *Fragmentation et CI contradictoires* : Pas d'accès aux données
- *En mise à jour* : Si violation alors ROLLBACK

Chapitre VI : Transactions et Accès concurrents

- Validation ou avortement sur chacun des sites impliqués
- Validations indépendantes ? NON car certaines transactions peuvent avoir avorté
- Un gestionnaire local par site
- *Modèle de système* :
  1. *Un gestionnaire des transactions* locales et distantes
  2. *Un coordonnateur de la gestion des transactions*

Gestion des transactions

- Chaque gestionnaire de transactions :
  - *Gère* le Log
  - *Participe à la coordination* des exécutions selon un protocole
- Chaque coordonnateur :
  - *Lance* l'exécution d'une transaction
  - *Décompose* une transaction
  - *Transfère* les transactions vers les "bons" sites
  - *Coordonne* la validation

Gestion des transactions : Plan

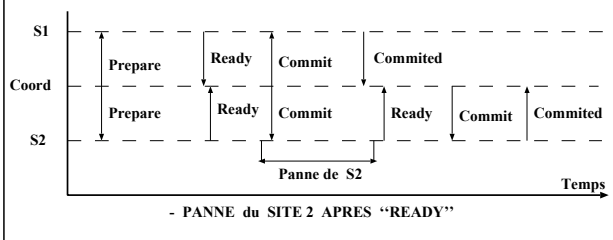
La validation des transactions

- *Protocole de validation à deux phases (Two-Phase Commit Protocol)* :
  - Lampson 1976, Gray 1978
  - Le plus connu et implémenté
  - Exécuté par chaque site
  - Contrôlé par un site maître (coordonnateur)
  - *Assure l'**A**tomicité* (sauf dans le cas de la destruction du log)
  - *Correction prouvée* dans "J.L. Baer and al.: *"The Two-Step Commitment Protocol: Modeling, Specification and Proof Methodology"*, ICSE'81, San Diego.

Le protocole de validation à deux phases

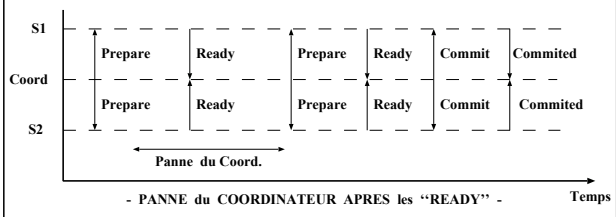
1. *Préparation* : Coordonnateur demande aux sites de se préparer (Ecriture Logs coordonnateur et sites, ...)
2. *Validation/Annulation* : Coordonnateur ordonne la validation ou l'avortement en fonction des résultats de la première phase

Validation à deux phases et échecs



- PANNE du SITE 2 APRES "READY"

Validation à deux phases et échecs



Validation à deux phases

- Transaction T initiée sur site S_i
- C_i coordonnateur de S_i
- Chaque site S_j participant à T informe C_i de la fin de sa "contribution"

1. *Phase 1* :
   - C_i :
     - Journalise < PREPARE T >
     - Envoie < PREPARE_TO_COMMIT > aux sites S_j
   - Chaque S_j :
     - Si validation possible
       Alors Journalise et envoie < PRET T > à C_i
     - SINON Journalise < NO T > et envoie < ABORT T > à C_i

Validation à deux phases

2. *Phase 2* : CAS REUSSITE (tous les < PRET T > sont parvenus)
   - Journalisation de < COMMIT T > sur S_i et les S_j
   - Les S_j envoient un "ACK" à C_i
   - C_i journalise < COMPLETE T> après réception des ACK.

3. *Phase 2* : CAS ECHEC
   - Un des S_j répond < ABORT T >
   - ou tous les < PRET T > non parvenus à C_i dans un délai d
   - C_i :
     - Journalise < ABORT T >
     - Envoie < ABORT T > aux S_j

Validation à deux phases

Exercice :
Décrire le protocole de validation à deux phases sous la forme d'un diagramme d'interaction, sachant que tous les sites utilisent le *Write Ahead Log Protocol*.

Validation à deux phases et pannes

1. Panne d'un site participant
2. Panne du site du coordinateur
3. Rupture de liaison
4. Partition du réseau

1/4. Validation à deux phases : Panne d'un site S_P

- *Après REPRISE, examen du Log de S_P* : Cas
  1. Pas d'information sur T dans le Log : S_P n'a pas répondu au $< PREPARE\ T >$ $\Longrightarrow$ UNDO(T)
  2. $< COMMIT\ T > \in Log$: REDO(T)
  3. $< ABORT\ T > \in Log$: UNDO(T)
  4. $< PRET\ T > \in Log$:
     - Consulter C_i puis UNDO(T) ou REDO(T)
     - *Cas S_i en Panne : Comment obtenir des informations sur T ?*
       * Envoi de $< QUERY\ STATUS >$ aux autres S_j
       * Si réception d'une réponse : UNDO ou REDO
       * Si pas de réponse ou si aucun S_j n'a l'information, poursuite des envois de $< QUERY\ STATUS >$ par S_P (avec état $< PRET\ T >$ et verrous éventuels "chez" S_P)

2/4. Validation à deux phases et Panne du coordinateur

- T est validée ssi $\exists\ S_j$ actif $\wedge\ <\ COMMIT\ T\ >\ \in\ Log\ de\ S_j$
- T est annulée ssi $\exists\ S_j$ actif $\wedge\ <\ NO\ T\ >\ \in\ Log\ de\ S_j$
- Sinon attendre la reprise de C_i

3/4. Validation à deux phases et Rupture d'une liaison

- Messages ne parviennent plus
- Chaque "partie" pense que l'autre est en panne
- Application du schéma "Panne d'un site"

4/4. Validation à deux phases et Partition du réseau

- Si S_i et tous les S_j dans la même partition : Pas de problème
- Sinon cas similaire à la rupture d'une liaison

Validation à deux phases : Implantations

1. Centralisé (cf. ce qui précède)
2. Linéaire
3. Distribué

Validation à deux phases : Mise en œuvre pratique

- Une transaction "maîtresse" (coordonateur) connaissant la fragmentation
- Une transaction par sous-requête
- Transaction "maîtresse" :
  - lance les sous-requêtes ($\simeq$ appel de procédure distante)
  - Coordonne la validation à deux phases
- Communication par appel de procédures distantes (*Remote Procedure Call*)

Validation à deux phases : conclusion

- Bloquant si panne du coordonnateur : Maintien des ressources jusqu'à sa reprise
- Blocage dû à la transition directe PRET, VALIDER
- *Protocole à trois phases* (Implantation ?)
  - Introduit par D. Skeen
    "Non-Blocking Protocols", ACM-SIGMOD Conference, An Arbor, Michigan, May 1981
  - Etat intermédiaire < PRET à VALIDER >
  - Si atteint alors VALIDER TOUJOURS
  - C_i en panne et aucun S_j n'a reçu < PRET à VALIDER > alors UNDO

Gestion de transactions : Plan

I. Les techniques d'estampillage

- Ordre de sérialisation obtenu par association d'une estampille unique à chaque transaction
- Gestion centralisée ou distribuée des estampilles
  1. *Gestion centralisée*
     - UN site
     - Compteur logique ou horloge locale
  2. *Gestion décentralisée*
     - < Estampille, Identifiant_du_Site >
     - Estampille : compteur logique ou fonction de l'horloge locale
     - *SYNCHRONISATION des HORLOGES ou des COMPTEURS*

Estampillage

- *SYNCHRONISATION des HORLOGES ou des COMPTEURS*
  - Problème : Un site génère plus rapidement que les autres
  - Mécanisme de contrôle de la génération "saine" des estampilles
    - Site S_i : Horloge H_i
    - S_i reçoit un message estampillé < t, s> avec $t > Valeur(H_i)$
    - S_i "avance" H_i à t + 1
  - Mécanisme similaire si utilisation d'horloges physiques

Estampilles et accès concurrents

- Chaque granule garde trace de la dernière transaction qui l'a utilisé
- Ordonnancement :

  T_i veut accéder à un granule estampillé j
  Si j $\leq$ i
  Alors T_i peut s'exécuter
  Sinon ABORT(T_i) et reprise ultérieure
  avec estampille $e > j$
  Finsi
- *ESTAMPILLAGE et ROLLBACK*
  - Conflits gérés par des ROLLBACK et pas par des attentes
  - Donc, risque de cascades de rollback

Estampillage et accès concurrents : Conclusion

- Beaucoup de ROLLBACK et de UNDO
- Mélange de 2-Phase Commit et estampillage : Protocole assurant la sérialisabilité sans cascade de ROLLBACK
- Technique délaissée au profit du verrouillage (+ gestion de deadlock)

II. Accès concurrents : Techniques de verrouillage

1. Données non dupliquées
2. Protocole de la majorité
3. Coordonnateur unique
4. Protocole "partial"
5. Protocole de la copie primaire

1/5. Verrouillage et données non dupliquées

- Un Gestionnaire de verrous GV_i par site S_i
- Demande de verrou d'un granule D, sur un site S_i par une transaction T :
  - T envoie sa demande à GV_i
  - D verrouillé dans un mode incompartible avec celui demandé $\Longrightarrow$ Demande mise en attente
  - Verrou apposé $\Longrightarrow$ GV_i informe T
- Mécanisme *SIMPLE* : Deux envois de messages
  - Demandes de verrouillage et de déverrouillage
- Gestion *COMPLEXE* du deadlock

2/5. Verrouillage et protocole de la majorité

- Version modifiée du protocole sans duplication
- D est détenu par une transaction T si T obtient le verrou sur une majorité de copies
- Un Gestionnaire de verrous GV_i par site S_i
- Granule D localisé sur N sites
- Demande concernant D adressée à M sites, M $>$ N/2
- Verrouillage ou attente sur les M sites
  - **(+)** Pas de contrôle centralisé avec duplication
  - **(-)** Gestion complexe du deadlock
  - **(-)** 2(n/2 + 1) messages (Verrouillage)
  - **(-)** + (n/2 + 1) messages (Déverrouillage)

3/5. Verrouillage et Coordonnateur unique

- Sur un site S_C
- Reçoit les demandes de (dé)verrouillage
- Lecture sur tout site détenant une copie
- Ecriture : concerne tout site détenant une copie
- Si verrou apposable : informer le demandeur
- Sinon : informer et mettre en attente
- *APPROCHE SIMPLE* :
  - Trois transferts de messages : 2 (Verrouillage) + 1 (Déverrouillage)
  - Gestion du deadlock : cf. BDD centralisée

3/5. Verrouillage et Coordonnateur unique

- *INCONVENIENTS* :
  - Engorgement de S_C
  - Blocage en cas de panne
- *ALTERNATIVE* :
  - Coordonnateurs sur plusieurs sites
  - Distribution de la gestion des verrous
  - Chaque gestionnaire se charge d'un sous-ensemble de données
  - **(+)** Réduction engorgement
  - **(-)** Gestion plus complexe du deadlock

4/5. Verrouillage et Protocole "partial"

- Similaire au protocole de la majorité
- Favoriser les verrous partagés au détriment des verrous exclusifs
- Un gestionnaire de verrous GV_i par site S_i
- GV_i chargé des verrous sur les données locales
- *Verrou partagé* : demande adressée à un des sites détenant la donnée
- *Verrou exclusif* : demande adressée à tous les sites
- Mise en attente ou apposition
  - **(+)** "Overhead" inférieur à celui des autres protocoles
  - **(-)** Overhead néanmoins, en écriture
  - **(-)** Gestion complexe du deadlock

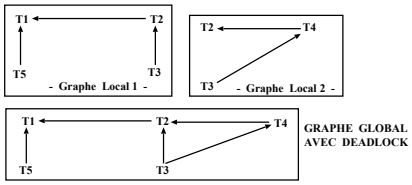
5/5. Verrouillage et Protocole de la copie primaire

- *SITE PRIMAIRE* : site privilégié d'une donnée dupliquée D
- Site primaire reçoit les demandes relatives à D
  - **(+)** Gestion de la concurrence similaire au cas non dupliqué
  - **(+)** Simplicité de l'implémentation
  - **(-)** Blocage en cas de panne sur le site primaire

III. Verrouillage et Interblocage

- *Cadre centralisé* :
  - Graphe d'attente
  - Scrutation périodique
  - Si $\exists$ cycle, tuer une transaction
- *Cadre distribué* : Gestion du graphe d'attente ?
  - Un graphe par site
  - $\neg\exists\ cycle\ local \not\Longleftrightarrow \neg\exists\ deadlock$

Verrouillage et Interblocage



- *Approches pour la gestion de l'interblocage* :
  1. Centralisée
  2. Répartie
  3. Hiérarchique

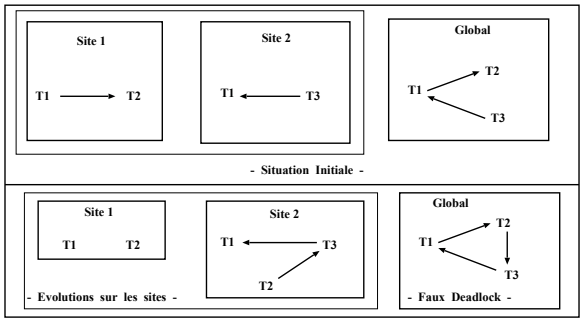
1/3. Gestion centralisée de l'interblocage

- COORDONNATEUR de détection de deadlock sur site S_C
- Graphe global géré sur S_C
- Communication des modifications des graphes locaux
- Mise à jour périodique du graphe global
  - Après chaque modification d'un graphe local
  - Après un certain nombre de modifications
  - Avant la scrutation

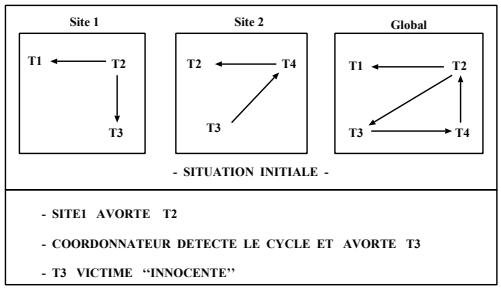
1/3. Gestion centralisée de l'interblocage

- *Graphe global construit* : Approximation du *graphe réel*
- Cycle sur le graphe construit
- Résolution (Tuer transaction T_D et ROLLBACK)
- Information aux sites impliqués dans T_D
- *!!! FAUX DEADLOCK et ROLLBACK INUTILE*

Exemple de faux deadlock



Exemple de Rollback inutile

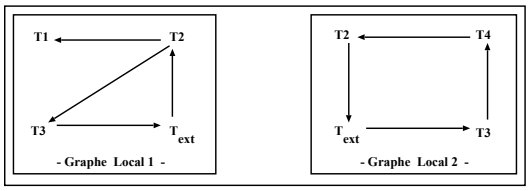


2/3. Gestion répartie de l'interblocage

- T_{ext} : Nouveau type de nœud (Transaction NON locale)
- Arc $T_i \longrightarrow T_{ext}$: T_i en attente d'un autre site
- Arc $T_{ext} \longrightarrow T_i$: T_{ext} en attente d'une ressource locale
- *Principe de la détection* :

  SI $\exists$ *cycle*
  ALORS SI $T_{ext} \notin$ *cycle*
  ALORS DEADLOCK LOCAL
  SINON - RISQUE de DEADLOCK
  - DETECTION [et RESOLUTION]
  FINSI
  FINSI

2/3. Gestion répartie de l'interblocage



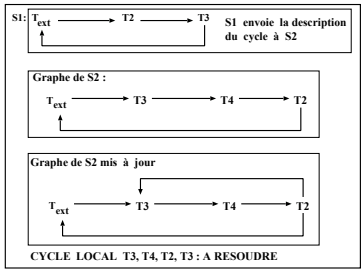
- Cycle nécessairement de la forme :

$$T_{ext} \longrightarrow T_{i1} \cdots \longrightarrow T_{ik} \longrightarrow T_{ext}$$

2/3. Gestion répartie de l'interblocage

- T_{ik} : Transaction k du site S_i
- S_i détecte le cycle : T_{ik} en attente de T_{ext} et *ext* = *j*
- S_i envoie un message à S_j :
  - Description du cycle
  - Demande de détection de deadlock
- S_j :
  - Mise à jour du graphe local
  - Recherche de cycle dans : graphe $\setminus \{T_{ext}\}$
  - $\exists$ cycle local : Résolution
  - $\exists$ cycle AVEC T_{ext} : Faire comme S_i
- *Nombre fini d'essais* : détection échoue ou réussit

2/3. Gestion répartie de l'interblocage : exemple



3/3. Gestion hiérarchisée de l'inter-blocage

- Hiérarchisation des sites
- Graphe d'attente d'un nœud : graphe global aux sites descendants
- Détection de deadlock sur les sous-arbres
- (+) Réduit l'activité à la racine (cf. approche centralisée)
- Performances fonction de l'adéquation *Hiérarchie-Localisation* des transactions

Chapitre VII : Sécurité de fonctionnement

Sécurité de fonctionnement et reprise

- Nouveaux types de pannes :
  - Panne d'un site
  - Rupture d'une liaison
  - Perte de message
  - Partition du réseau
- *Procédure* :
  1. *DETECTION*
  2. *RECONFIGURATION du SYSTEME*
  3. *REPRISE*

1/3. Sécurité de fonctionnement : Détection

- Souvent possible
- Identification de panne difficile
  - S_1 ne communique plus avec S_2
  - S_2 en panne ou rupture de liaison ?

2/3. Sécurité de fonctionnement : Reconfiguration

- Avortement des transactions actives (Libération des ressources)
- Inhiber l'accès aux données dupliquées du site en panne (MAJ catalogue)
- Si site en panne = site central (coordonnateur, etc.) : *Remplacement*
- Si partition du réseau :
  - Tolérer l'exécution de transactions dans les partitions
  - Eviter l'élection de plusieurs serveurs centraux dans une partition
  - Eviter la MAJ de données dupliquées par plusieurs partitions

3/3. Sécurité de fonctionnement : Reprise

- Appliquer les MAJ qui ont eu lieu sur les données dupliquées
- MAIS, ces données peuvent ENCORE être en cours de MAJ
- *Solution 1* :
  - Arrêt du système et remise en état
  - Eventuellement, négociation humaine pour la cohérence des copies
- *Solution 2* : Reprise $\simeq$ Série de transactions
- *Reprise après partition du réseau* :
  - Informer les sites de la réparation de la liaison (message broadcast)

Remplacement d'un site coordonnateur

- Coordonnateur UNIQUE $\Longrightarrow$ RISQUE de BLOCAGE
- Désignation d'un "remplaçant" :
  1. "Doublure" pré-définie (*BACK-UP*)
  2. Élection
  3. Préemption

1/3. Coordonnateur "back-up"

- Reçoit les mêmes messages que le "vrai"
- Exécute les mêmes algorithmes
- Gère les mêmes informations
- Prend le relai en cas de panne

(+) Reprise rapide
(-) Overhead

2/3. Élection de coordonnateur

- Identifiants UNIQUES des sites
- Panne du site de coordination :
  - Choix d'un site
  - Informer les autres sites
  - Prévoir d'informer les sites en panne
- Panne (présumée) du coordinateur : Pas d'écho pendant $\triangle$t $\Longrightarrow$ Élection

2/3. Élection de coordonnateur : Algorithme "bulldozer"

- *Algorithme* : S_i = initiateur de l'élection

  $S_i \longrightarrow$ Message d'élection vers les sites d'identifiants supérieurs
  SI pas de réponse APRES $\triangle$t
  ALORS • $\simeq$ "TOUS en PANNE"
  • S_i : "JE SUIS LE COORDINATEUR"
  • S_i informe tous les sites d'identifiants inférieurs
  SINON Attente élection d'un site d'identifiant supérieur
  SI PAS d'information à l'issue de l'attente
  ALORS • $\simeq$ PANNE du site ayant répondu
  • REFAIRE une ELECTION
  FINSI
  FINSI

3/3. Préemption

- Reprise d'un site S_i : Exécution du même algorithme

  SI $\neg\, \exists k, k = ID(Site_l) \wedge k > ID(S_i)$
  ALORS S_i DEVIENT COORDONNATEUR
  FINSI

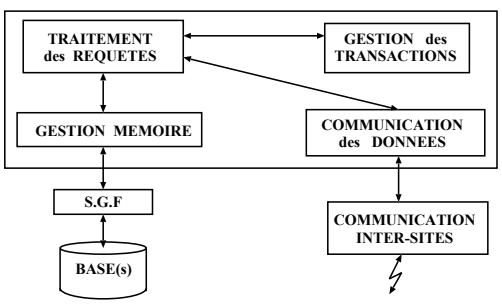
Chapitre VIII : Architecture des SGBD distribués

1. Homogènes
2. Hétérogènes
3. Systémes multi-bases
4. Bases de données parallèles
5. Architectures client/serveur

VIII.1- Bases de données distribuées homogènes

- R^*, successeur de SYSTEM-R, IBM, San José, 1979-1984
- Sites autonomes dotés du même SGBD
- Indépendance vis-à-vis de la localisation
- Transparence vis-à-vis de la duplication et de la fragmentation (ignorée pour simplifier l'implémentation)
- Extensions de SYSTEM-R
  - Définition de données et Dictionnaire
  - Traitement des requêtes
- Dictionnaire réparti
- Unicité des noms et immutabilité : Nom généré par le site qui crée un objet

Architecture de R^*



R^* : Traitement des requêtes

1. *Compilation de requêtes* :
   - *Site "client"* prend des décisions globales :
     - Choix des sites d'exécution (cf. jointures)
     - Choix de la méthode de transfert
     - Génération de plan
   - *Sites serveurs :*
     - Ordre de jointures
     - Plans d'exécution locaux

R^* : Traitement des requêtes

2. *Exécution* :
   - Un processus R^* par site inter-agissant avec un programme d'aplication
   - Processus vivant jusqu'à la fin du programme
   - Communication inter-sites :
     - Entre processus R^*
     - Totalement contrôlée par R^*

R^* : Gestion des transactions

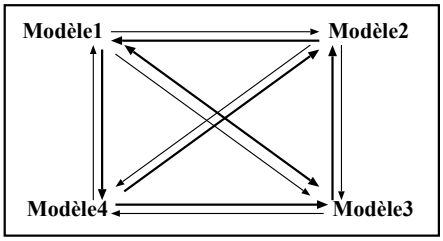
- 2-Phase Commit (Modifié)
- Détection répartie de deadlock
- Résolution de deadlock locaux et globaux

VIII.2- Bases de données distribuées hétérogènes

- Objectif : Indépendance, autonomie, transparence des SGBD locaux
- Intégration en une base globale de bases existantes sous des SGBD différents
- Problème difficile : certains systèmes n'admettent que des SGBD relationnels ou ne supportent que des requêtes d'interrogation
- Nouvelles fonctions : Correspondance/Traduction
  - Modèles (et représentation) de données
  - Langages de manipulation

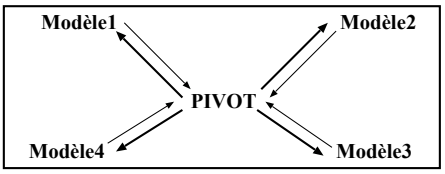
Intégration : approche 1 (approche "naïve")

- n*(n-1) traducteurs
- Difficulté de développement : (in)compatibilité des modèles



Intégration : approche 2 (modèle pivot ou commun)

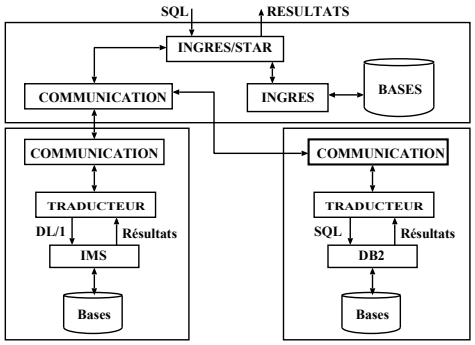
- Modèle pivot (Relationnel, Objet, (XML ?))
- Schéma global et LMD dans le modèle pivot
- Autant de traducteurs (bi-directionnels) que de modèles (2*n)
- Facilite la conception et la manipulation
- Dificulté d'intégration (conflits sémantiques, de nommage, etc.)



Exemples de systèmes

- ORACLE/STAR : SGBD réparti "relationnellement" hétérogène
- Sybase
- INGRES/STAR : SGBD relationnels et non relationnels (théoriquement)

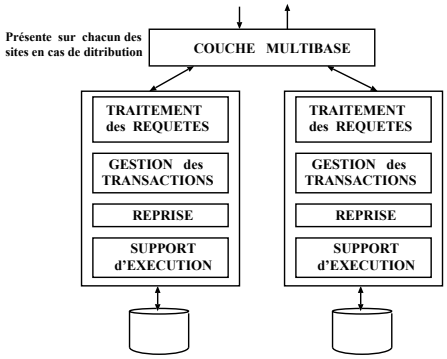
Architecture de Ingres/Star



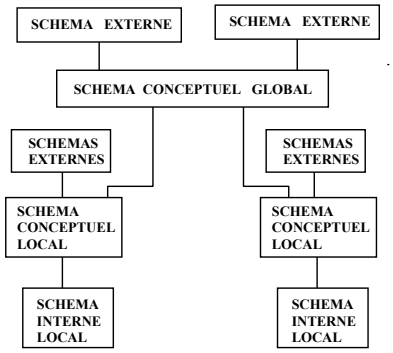
VIII.3- Multibases distribuées

- *SGBD logiquement intégrés distribués* :
  - Vision de toute la base
  - Base globale = ∪ Bases locales
- *SGBD multi-bases distribués* :
  - Fédération de bases de données
  - Vision de parties de bases locales que chaque SGBD veut partager
  - Base globale ⊆ ∪ Bases locales
  - Avec ou sans schéma conceptuel global

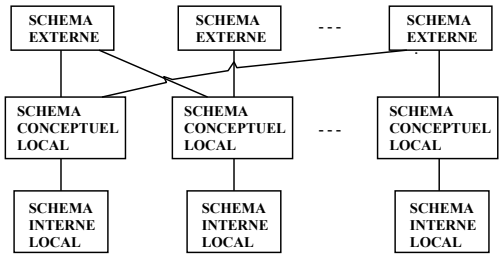
Composants fonctionnels des systèmes multi-bases



1/2. Multi-bases avec schéma global



2/2. Multi-bases sans schéma global



2/2. Multi-bases sans schéma global

- *Couche SGBD locaux + Couche système multibase*
- Accès aux bases délégué au mécanisme de correspondance entre schéma externe et schéma conceptuel local
- *Schéma des dépendances* inter-bases : expression de liens sémantiques
- *Extensions du LDD*
  - Manipulation de schémas (dépendances, externes, multibase)
    - CREATE, ALTER, DROP relation ou base
    - CREATE TABLE à partir de données d'une base privée
    - CREATE DATABASE : introduire une base dans la fédération
  - Nommage : *Nom_base•Nom_relation*
  - CREATE VIEW "multibase"
- *LMD* : SQL + Ouverture/Fermeture de plusieurs bases

Systèmes Multi-bases : Conclusion

- Technologie pas (encore) au point
- Oracle et Sybase offrent des possibilités de requêtes multi-bases centralisées
- Solutions techniques dans les SGBD répartis inadaptées aux multi-bases
- Problématique similaire dans la construction d'entrepôts de données (*Data Warehouse*)

VIII.4- Bases de données parallèles

- Bénéficier des architectures parallèles et multi-processeurs
- *Exemples* :
  - Machine transactionnelle *NonStop SQL*
  - Machine base de données DBC/1012 [Tera Data Corp.]
- Architectures multi-processeurs : Unités de traitement indépendantes coopérant via un réseau

Caractéristiques d'un SGBD parallèle

- Ressemble à un SGBD réparti homogène fortement intégré
  - Chaque nœud correspond à un site
- Diffère d'un SGBD réparti :
  - Spécialisation d'un nœud (gestion de données) $\Longrightarrow$ implantation plus simple et plus efficace
  - Nombre de nœuds peut être très élevé (Jusqu'à 1024 processeurs à MIPS dans DBC/1012)

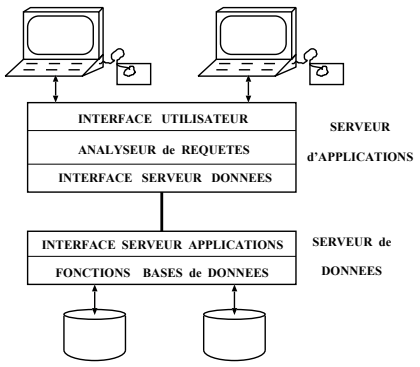
Caractéristiques d'un SGBD parallèle

- *Exemple* : Sybase MPP (Sybase11, 1995)
  - 128 processurs ou plus
  - Sun-Sparc Center 2000E, 16 processurs :
    - 4544 tpmc; 26 % > Informix
  - HP 9000 T500, 12 processurs :
    - 5621 tpmc; 300 % > Oracle

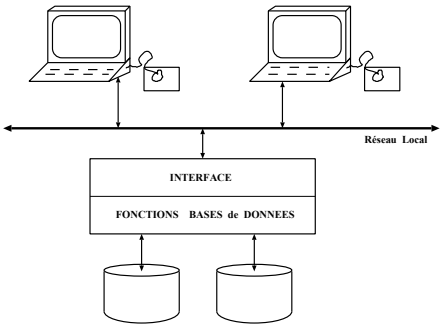
VIII.5- Architectures client-serveur

- Stations de travail et machines parallèles
- *Serveur d'applications* : station exécutant des programmes (anciennement *machine hôte*)
- *Serveur de données* : machine dédiée (anciennement *machine bases de données, back-end*)
- *Architectures* :
  1. Serveur de données centralisé
  2. Serveur distribués

Architecture client-serveur



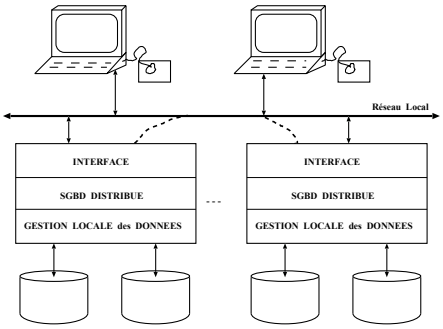
1/2. Architecture centralisée



Architecture centralisée

- Alternative moins coûteuse que SGBD distribué
- Accès efficace au serveur de données
- Mêmes inconvénients qu'une architecture "classique"
- Informations du dictionnaire en lecture seule sur les stations
  - Tâches locales de contrôle (autorisations, vues, etc.)
  - Réduction du flux de communication avec le serveur de données
- Serveur de données mono/multi-processeur(s)

2.2/ Architecture Distribuée



2.2/ Architecture Distribuée

- Serveurs d'applications peuvent être similaires à ceux de l'architecture centralisée
  - Requête addressée au serveur local,
  - Serveur local se charge de l'exécution répartie
- Alternative plus complexe : Dictionnaire et traitements de requêtes distribués sur chaque serveur d'applications

Chapitre IX : Conclusion

Conclusion

- Système distribué : {Sites} gérant des bases locales
- AUTONOMIE, TRANSPARENCE et COOPERATION
- ROBUSTESSE :
  - Détection panne
  - Reconfiguration (Tolérance au fonctionnement en mode dégradé)
  - Reprise après panne
  - Remplacement du coordonnateur
- ATOMICITE des transactions PLUS COMPLEXE (2PC)
- ADAPTATION des mécanismes "classiques" à la gestion des accès concurrents (Verrous, estampilles)

Conclusion

- Gestion centralisée, distribuée, hiérarchisée du deadlock
- Graphes d'attente locaux, globaux

- *Technologie trop complexe ?*
- % architecture client/serveur et outils de connectivité (ODBC, JDBC) ?
- Le Web comme une base de données distribuée ?

References

[1] Abiteboul (S.), Buneman (P.) et Suciu (D.). – *Data on the Web; From Relations to Semi-Structured Data and XML*. – San Francisco, CA, Morgan Kaufmann Publishers, 2000.

[2] Agrawal (R.) and al.. – *The Claremon Report on Database Research*. – CACM, 52(6), June 2009. http://doi.acm.org/10.1145/1516046.1516062.

[3] ANSI/X3/SPARC. – Study Group on Data Base Management Systems. – Interim Report - ACM, 1975.

[4] Besancenot (J.) et al. – *Les systèmes transactionnels: concepts, normes et prouits*. – Paris, Editions Hermes, 1997, *Collection informatique*.

[5] Boudjlida (N.). – Tutoriel "Objets Distribués, Intéropérabilité, CORBA" (*Distributed Objects, Interoperability, CORBA : a tutorial*). *In : Journées Bases de Données Avancées, BDA'98*. – Hammamet, Tunisie, Octobre 1998. (in French, http://www.loria.fr/~ nacer).

[6] Boudjlida (N.). – *Bases de données et systèmes d'informations. Le modèle relationnel: langages, systèmes et méthodes*. – Dunod, Paris, 1999. Cours et exercices corrigés. Collection Sciences Sup.

[7] Boudjlida (N.). – De la technologie bases de données aux technologies web. – Conférence invitée, 7th Maghrebian Conference on Software Engineering and Artificial Intelligence, MCSEAI'2002, Annaba, DZ, May 2002. http://www.loria.fr/~ nacer/PUBLI/Tut-MCSEAI02.ZIP.

[8] Boudjlida (N.). – *Gestion et Administration des Bases de Données: Application à Sybase et Oracle*. – Dunod, Paris, 2003.

[9] Boudjlida (N.) et Belhamissi (Y.). – Traitement de requêtes réparties : des modèles à leur implémentation. *In : Actes des Journées Internationales des Sciences Informatiques, JISI'94*. – Tunis, Tunisie, Mai 1994.

[10] Bouneffa (M.A.) et Boudjlida (N.). – Managing Schema Changes in Object-Relationship Databases. *In : Proceedings of the 14th Object-Oriented and Entity-Relationship International Conference, OO-ER'95*, éd. par Papazoglou (M.P.). pp. 113–122. – Gold Coast, Australia, December 1995.

[11] Elmasri (R.) et Navathe (S.B.). – *Fundamentals of database systems*. – The Benjamin/Cummings Publishing Company, Inc., 1989.

[12] Gardarin (G.) et Gardarin (O.). – *Le Client-Serveur*. – Paris, Eyrolles, 1996.

[13] Oszu (M. Tamer) et Valduriez (P.). – *Principles of Distributed Database Systems*. – Prentice Hall International, December 1998. 2nd edition.

Systèmes de Gestion des Bases de Données

Traitement des requêtes dans les SGBD-R

Nacer Boudjlida

Traitement des requêtes dans les SGBD relationnels

- Evaluation des requêtes :
  - Données de la base vers la mémoire centrale
  - Traitement
  - "Retour" dans la base si mise à jour
- Peut nécessiter la création de tables intermédiaires
- Volume des transferts et des tables de travail fonction des tailles des relations mises en jeu
- *Objectif d'un optimiseur* : Réduction des volumes et du temps

Traitement des requêtes : Besoins d'optimiser ?

- Temps d'exécution d'une requête "anormal" % taille des relations, existence d'index
- Une requête s'exécute plus lentement que des requêtes similaires
- ↗ temps d'exécution d'une requête
- Temps d'exécution d'une requête en tant que procédure > celui de son exécution en tant que requête
- Plan d'exécution utilisant un parcours de relation au lieu d'utiliser un index

Traitement des requêtes : Origines des mauvaises performances ?

- "Vieilles" statistiques sur la distribution des données
- Inexistence d'index appropriés
- Index en cours d'utilisation pour accéder à une table volumineuse
- Clause *where* conduisant au choix d'une mauvaise stratégie
- Procédure non re-compilée après changements significatifs
- etc.

Traitement des requêtes : Les méthodes

1. Syntaxique :
   - Fondement : Heuristiques + Propriétés de l'algèbre
   - Transformation d'arbres (algèbre) ou de graphes (calcul relationnel)
2. Estimation de coûts d'exécution de diverses stratégies d'évaluation
3. Sémantique : Exploitation des contraintes d'intégrité

- Méthodes non exclusives : 1 et 2 souvent combinées

Traitement des requêtes : Plan

1. Transformation d'arbres algériques
2. Transformation de graphes de requêtes
3. Optimisation à base de coûts
4. Optimisation Sémantique
5. Processus Général d'Optimisation

I. Transformation d'arbres algériques

- Heuristiques : $\searrow$ taille des opérandes des opéraions les plus coûteuses
  1. "Descente" des sélections : $\searrow$ les relations "en hauteur"
  2. "Descente" des projections : $\searrow$ les relations "en largeur"
- [Phrase SQL $\longrightarrow$] Arbre algébrique :
  - *Feuilles* : Relations
  - *Racine* : Résultat de la requête
  - *Nœuds internes* : Opérations de l'algèbre
  - *Arcs "entrants"* : Opérandes
  - *Arcs "Sortants"* : Résultat de l'évaluation d'un nœud
- Evaluation de "bas en haut" (gauche-droite ou droite-gauche)

I. Transformation d'arbres algériques : Exemple

- *Personne*(id#, nom, prenom, date_naisssance, adresse, equ#)
- *Equipe*(equ#, nom_equ, resp_equ#)
- *Projet*(proj#, nom_proj, lieu_proj, equ#)
- *Travaille_sur*(id#, proj#, taux)
  - *Personne.equ#* : Equipe de rattachement
  - *Projet.equ#* : Equipe en charge du projet
  - *Travaille_sur.taux* : Part de temps sur un projet
- *Requête* :
  Personnes nées après 1962 et travaillant sur le projet "BDD"

I. Transformation d'arbres algériques : Exemple

- En SQL :

```
SELECT  nom
FROM    Projet, Travaille_sur, Personne
WHERE   Projet.nom_proj = 'BDD'
AND     Personne.id# = Travaille_sur.id#
AND     Personne.date_naissance > 'dec-31-1962'
AND     Travaille_sur.proj# = Projet.proj#
```

- *Représentation dite canonique* :
  - *Relations de la clause FROM* $\longrightarrow$ Produit cartésien (X)
  - *WHERE* $\longrightarrow$ Sélection (σ), nœud père du sous-arbre X
  - *SELECT* $\longrightarrow$ Projection (Π) en racine de l'arbre

Transformation d'arbres algériques : Arbre initial (0)

Transformation d'arbres : Arbre 1

- *"Descente" des sélections*

Transformation d'arbres : Arbre 2

- *Permutation de Personne et de Projet* (cf. notion de sélectivité) : **Si mélange méthodes syntaxique et estimation de coûts**.

Transformation d'arbres : Arbre 3

- *Introduction de jointures*

Transformation d'arbres : Arbre 4

- *Introduction de projections* : ne "faire remonter" que les attributs utiles

Transformation d'arbres : Sur l'arbre initial (Arbre 0)

- *Projet* : 20 tuples de 100 caractères
- *Travaille_sur* : 100 tuples de 50 caractères
- *Personne* : 500 tuples de 100 caractères
- *Deux produits cartésiens* : 10^6 tuples de 250 caractères !

Transformation d'arbres : Sur l'arbre final (Arbre 4)

- *1ère jointure* :
  - Une relation à une colonne et probalement à un n-uplet
  - Relation *Travaille_sur* : deux colonnes
- *2ème jointure* :
  - Une relation à une colonne (sous-arbre gauche)
  - Une relation à 2 colonnes (sous-arbre droit) réduite aux seuls n-uplets satisfaisant la sélection

Transformation d'arbres : Règles de transformation

- ***(R1)*** Décomposition des expressions de sélection.
  $Select(R,\ C_1 \wedge C_2 \wedge \cdots \wedge C_{n-1} \wedge C_n) \longrightarrow$
  $Select(Select(\cdots\ Select\ (Select(R,\ C_n),\ C_{n-1})),\ \cdots),\ C_2),\ C_1)$
- ***(R2)*** Commutativité de la sélection.
  $Select(Select(R,\ C_2), C_1) \longrightarrow Select(Select(R,\ C_1),\ C_2)$.
- ***(R3)*** Restriction de la liste de projections.
  $Proj_{L1}(Proj_{L2}(\cdots\ (Proj_{Ln}(R))\ \cdots) \longrightarrow Proj_{L1}(R)$.
- ***(R4)*** Commutation de la sélection et de la projection.
  Si C ne porte que sur les A_i :
  $Proj_L(Select(R,\ C)) \longrightarrow Select(Proj_L(R),\ C)$

Transformation d'arbres : Règles de transformation

- ***(R5)*** Commutativité de la jointure (et du produit cartésien).
  $Join_C\ (R,\ S) \longrightarrow Join_C(S,\ R)$.
- ***(R6)*** Commutation sélection-jointure (ou produit cartésien).
  - ***(R61)*** Si les attributs de la condition C de sélection n'appartiennent qu'à une des relations de la jointure, par exemple R, alors :
    $Select(Join_E(R,\ S),\ C) \longrightarrow Join_E(Select(R,\ C),\ S)$.
  - ***(R62)*** Sinon, si la condition C peut se ré-écrire en $C_1 \wedge C_2$ où :
    - C_1 ne porte que sur des attributs de R
    - $C_1\ C_2$ ne porte que sur ceux de S, alors :
      $Select(Join_E(R,\ S),\ C) \longrightarrow$
      $Join_E(Select(R,\ C_1),\ Select(S,\ C_2))$

Transformation d'arbres : Règles de transformation

- ***(R7)*** Commutation projection-jointure (produit cartésien).
  - ***(R71)*** $Proj_L(Join_C(R,\ S)) \longrightarrow Join_C(Proj_{L_1}(R),\ Proj_{L_2}(S))$
    1. Si attributs de C = attributs de L
    2. Si $L = L1||L2$ avec $L_1 = Liste(A_i),\ A_i$: attributs de R et $L_2 = Liste(B_j),\ B_j$: attributs de S
  - ***(R72)*** Si la condition de jointure comporte des sous-listes L_3 d'attributs de R et L_4 d'attributs de S n'appartenant pas à L :
    1. Les ajouter aux projections "internes"
    2. Appliquer une projection "externe" sur L

    $Proj_L(Join_C(R,\ S)) \longrightarrow Proj_L(Join_C(Proj_{L_1,\ L_3}(R), Proj_{L_2,\ L_4}(S)))$.

Transformation d'arbres : Règles de transformation

- ***(R8)*** Commutativité des opérations ensemblistes ($\cup$, $\cap$).
- ***(R9)*** Associativité : jointure, produit cartésien, union et intersection. $((R\ op_i\ S)\ op_j\ T) \longrightarrow (R\ op_j (S\ op_i\ S))$ où R, S, T sont des relations et op_i, op_j des opérateurs parmi ceux cités
- ***(R10)*** Commutation de la sélection et des opérations ensemblistes. (Select((R op S), C)) $\longrightarrow$ (Select(R, C) op Select (S, C)) où op est l'opérateur d'union, d'intersection ou de différence

Transformation d'arbres : Règles de transformation

- ***(R11)*** Commutation de la projection et des opérations ensemblistes. $(Proj_L(R\ op\ S)) \longrightarrow ((Proj_L\ (R))\ op\ (Proj_L(S)))$.
- Autres :
  - Equivalences logiques
  - Autres propriétés des opérateurs algébriques

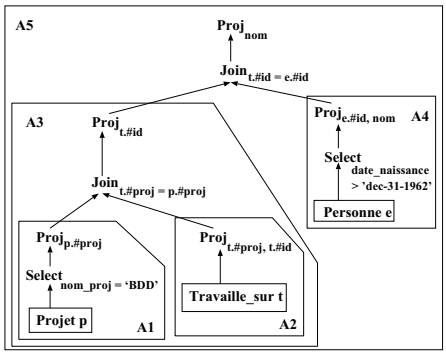
Transformation d'arbres : Ebauche d'algorithme

1. Transformer toute expression de sélection conjonctive en une "cascade" de sélections (Règle R1)
2. "Descendre" les sélections le plus bas possible ((R2), (R4), (R6), (R10)).
3. Ordonner les feuilles de l'arbre de sorte que les sélections les plus restrictives soient évaluées en premier ((R8), (R9))
   - **Etape à n'exécuter que si** combinaison méthode syntaxique et méthode par estimation de coûts ;
   - "Plus restrictives" = produisant les plus petites relations ;
   - cf. distribution des valeurs, index (dictionnaire) ;
   - cf. notion de sélectivité, plus loin.

Transformation d'arbres : Ebauche d'algorithme

4. (Produit cartésien; Sélection) $\longrightarrow$ Jointure, où la condition de jointure est la condition de sélection.
5. "Fragmenter" et "faire descendre" le plus bas possible les listes de projection, en créant de nouvelles projections, si besoin est (Règles (R3), (R4), (R7), (R11)) ,
6. Identifier et ordonner les sous-arbres qui représentent des groupes d'opérations pouvant être exécutés par une seule routine du SGBD.

Transformation d'arbres : Exemples de plan (Etape 6)



Traitement des requêtes : Plan

1. Transformation d'arbres algériques
2. Transformation de graphes de requêtes
3. Optimisation à base de coûts
4. Optimisation Sémantique
5. Processus Général d'optimisation

II. Optimisation de graphes de requêtes

- Technique dite de décomposition de requêtes
- Introduite dans Ingres/QUEL (Variables n-uplets)
- *Graphe de requêtes* :
  - *Sur les arcs* : Conditions de jointure des nœuds reliés
  - *Nœuds* : Variables de tuples ou constantes de la requête
  - *Arcs "'conditions de sélection"'* : relient des nœuds de constantes aux nœuds des variables impliquées dans les conditions.
- Requête à n variables $\longrightarrow$ m requêtes, plus simples, à (n-1) variables.

Traitement des requêtes : Plan

1. Transformation d'arbres algériques
2. Transformation de graphes de requêtes
3. Optimisation à base de coûts
4. Optimisation Sémantique
5. Processus Général d'Optimisation

III. Optimisation par estimation de coûts

- Minimiser une fonction côut, avec comme facteurs :
  - Accès disque,
  - Coût du traitement en mémoire centrale,
  - Taille des n-uplets, des relations,
  - Index : Existence, nombre de niveaux,
  - Coût de communication (requête, résultats) entre sites (architectures client/serveur, par exemple),
  - etc.
- Utilisation effective (souvent) des coûts des accès

III. Optimisation par estimation de coûts

- *Informations du dictionnaire*
  - Nombre exact ou estimé de tuples et de blocs physiques par relation ;
  - Nombre de niveaux d'index ;
  - Nombre de valeurs distinctes par attribut, dans les cas où un index ou un cluster existe : permet d'estimer le nombre moyen d'enregistrements qui satisfont une sélection (*sélectivité* ou *cardinal de sélection* d'un attribut A).
    - A est clé : *s = 1*
    - A n'est pas clé : *s = Card(R)/Nombre de valeurs de A.*

III. Optimisation par estimation de coûts

- *Sélectivité de la jointure* : $sj = Card((R \bowtie_{Cond} S))/Card(R \times S)$
  - *sj* = 1 si pas de condition de jointure
  - *sj* = 0 si aucun tuple ne vérifie *Cond.*
- *Cas de l'équi-jointure* : (*Condition du type R.A = S.B*) Cas particuliers :
  1. A est clé de R : $Card(R \bowtie S) \leq Card(S)$
     - $sj \leq Card(S)/Card(R) \times Card(S)$
     - $sj \leq 1/Card(R)$
  2. De même, si B est clé de S : $sj \leq 1/Card(S)$.
- D'où la taille estimée de $R \bowtie S$: *sj* $\times$ *Card(R)* $\times$ *Card(S)*.

Exemples d'estimation de l'équi-jointure

- br : nombre de blocs de R,
- bs : nombre de blocs de S
- k : facteur de blocage de la relation résultat

1. *Jointure par boucles imbriquées*
   - R dans la boucle extérieure ; *sj* donné ; 2 buffers
   - Coût estimé : *br + (br * bs) + ((sj * Cardinal(R) * Cardinal(S))/k)*
   - Dernier facteur : Coût de l'écriture du résultat.

Exemples d'estimation de l'équi-jointure

2. *Jointure par tri-fusion*
   - Si relations déjà triées : *Coût* = $br + bs$
   - Sinon, *Coût* = $br + bs + l * (br * \log_2 br + bs * \log_2 bs)$.
     - $(l * b * \log_2 b)$: Approximation du tri
     - b : nombre de blocs
     - l : facteur constant.

IV. Optimisation sémantique

- *Contrainte d'intégrité* Formule logique CI
- *Expression de sélection* E
- *Pas d'accès à la base si* $\neg(E \wedge CI)$
- *Exemple* :
  - CI : Tous les vols long courrier partent de Paris-Roissy
  - E : Vols long courrier partant de Paris-Orly ?
- Méthode non implantée (démonstration automatique)

Optimisation de requêtes : SGBD Sybase

- Génération de plans avec/sans exécution de requête
- Exemple : Visualisation d'un plan sans exécution

  SET SHOWPLAN ON
  SET NOEXEC ON
  Requête dont on veut examiner le plan.
- Plans d'exécution et procédures stockées :
  - Exécution avec *WITH RECOMPILE* : MàJ du plan stocké
  - Création avec *WITH RECOMPILE* : Génération systématique
- *Mise à jour* des informations sur la distribution des valeurs des clés des index :

  UPDATE STATISTICS NomDeRelation [NomIndex]

Optimisation de requêtes : SGBD Oracle

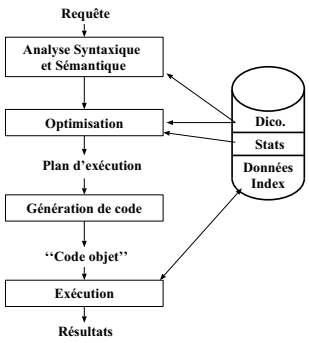
- Commande *EXPLAIN PLAN* : obtenir le plan d'exécution
- Plan rangé dans *PLAN_TABLE* ou dans une relation créée préalablement à l'exécution de *EXPLAIN PLAN* (clause *INTO* de *EXPLAIN PLAN*)

  EXPLAIN PLAN
  [*SET STATEMENT_ID* = identification_plan]
  [*INTO* [nom_utilisateur.]nom_de_relation] *FOR* requête_SQL
- ANALYZE {index| table | cluster} {COMPUTE | ESTIMATE} STATISTICS

Traitement des requêtes : Plan

1. Transformation d'arbres algériques
2. Transformation de graphes de requêtes
3. Optimisation à base de coûts
4. Optimisation Sémantique
5. Processus Général d'Optimisation

V. Optimisation : Processus général



Traitement des requêtes : Processus général

1. Normalisation des prédicats
2. Analyse sémantique (Requête = Graphe connexe)
3. Simplification des formules logiques
4. Mise sous forme d'arbre relationnel

1/4) Normalisation des prédicats

- *Forme normale conjonctive* : conjonction de disjonctions
  $(p_1 \vee p_2 \vee \cdots \vee p_n) \wedge \cdots \wedge (q_1 \vee \cdots \vee q_m)$
- *Forme normale disjonctive* : disjonction de conjonctions
  $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \vee \cdots \vee (q_1 \wedge \cdots \wedge q_m)$
- *Formules sans quantificateurs* :
  - Commutativité, Associativité de la conjonction et de la disjonction
  - Distributivité $(\vee, \wedge)$, $(\wedge, \vee)$
  - $\neg(p_1 \wedge p_2) \longleftrightarrow \neg p_1 \vee \neg p_2$
  - $\neg(p_1 \vee p_2) \longleftrightarrow \neg p_1 \wedge \neg p_2$
  - $\neg(\neg p) \longleftrightarrow p$
- *Formules avec quantificateurs* : mise sous forme *prenex*

- *Forme disjonctive* :
  - Requête traitée comme une union de sous-requêtes
  - Risque de calcul redondant
- *Forme conjonctive* : généralement plus de "ET" que de "OU"
- *Exemple* :

  select libelle from produit p, stock s
  where p.prod# = s.prod# and s.adr = "Nancy"
  and (s.qte = 1000 or s.qte=200)

  - *Forme disjonctive* :

    (p.prod# = s.prod# $\wedge$ s.adr = "Nancy" $\wedge$ s.qte = 1000)
    $\vee$ (p.prod# = s.prod# $\wedge$ s.adr = "Nancy" $\wedge$ s.qte = 2000)
  - *Forme conjonctive* :

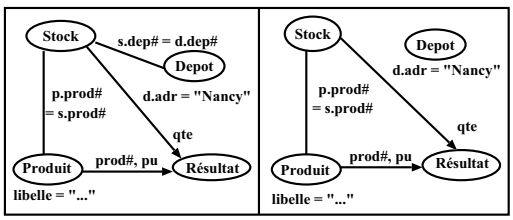
    p.prod# = s.prod# $\wedge$ s.adr = "Nancy" $\wedge$ (s.qte = 1000 $\vee$ s.qte=2000)

2/4) Analyse

- *Dictionnaire* : Relations, attributs connus
- *Typage des expressions*
- *Correction sémantique* : pas de sous-requête isolée
  - *Calcul relationnel* : impossible à determiner
  - *Requêtes sans $\vee$ ni $\neg$* : graphe connexe
- *Graphe de requête* :
  - *Nœud* : résultat ou opérande
  - *Arcs entre nœuds non résultats* : Jointure
  - *Arcs d'extrêmité nœud résultat* : Projection
  - *Nœud non résultat* peut être labellé par une expression de sélection ou une auto-jointure

- *Exemple* :

  select p.prod#, p.pu, s.qte
  from produit p, stock s, depot d
  where p. prod# = s.prod# and s.dep# = d.dep#
  and s.qte >= 0 and d.adr = "Nancy" and libelle = "..."



3/4) Simplification d'expressions logiques

- *Cas de l'utilisation de vues*

  create view V
  as select * from produit
  where pu $>$ 100.0 and pu $<$ 200.0

- *Requête* :

  select * from V
  where pu $<$ 200.0

- *Après ré-écriture* :

  select * from produit
  where pu $>$ 100.0
  and pu $<$ 200.0 and pu $<$ 200.0

3/4) Simplification d'expressions logiques (suite)

- *Elimination de la redondance* : règles d'idempotence
  - $p \wedge p \longleftrightarrow p; p \vee p \longleftrightarrow p$
  - $p \wedge Vrai \longleftrightarrow p; p \wedge Faux \longleftrightarrow Faux$
  - $p \vee Vrai \longleftrightarrow Vrai; p \vee Faux \longleftrightarrow p$
  - $p \wedge \neg p \longleftrightarrow Faux; p \vee \neg p \longleftrightarrow Vrai$
  - $p_1 \wedge (p_1 \vee p_2) \longleftrightarrow p_1$
  - $p_1 \vee (p_1 \wedge p_2) \longleftrightarrow p_1$

3/4) Simplification d'expressions logiques (suite et fin)

- *Exemple* :

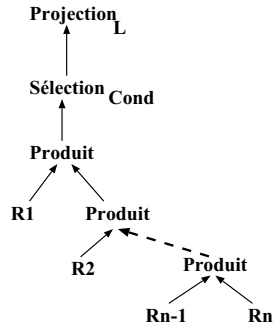
  select libelle from produit
  where NOT libelle = "K7Cr"
  and (libelle = "K7Cr" or pu = 20.0)
  and NOT pu = 20.0
  and prod# = 4

  se simplifie en :

  select libelle from produit
  where prod# = 4

4/4) Mise sous forme d'arbre algébrique

SELECT L
FROM R1, R2, ..., Rn
WHERE Cond



Systèmes de Gestion des Bases de Données

Accès concurrents et Sécurité de fonctionnement

Nacer Boudjlida

Introduction à la notion de transaction

- Partage des ressources dans les systèmes d'exploitation
- Base de données accédée par plusieurs utilisateurs (ou programmes)
- *Sous-système de gestion des accès concurrents*
- *Techniques* : Verrouillage, exclusion mutuelle
- *Ressources partagées* : Données et méta-données
- *Transaction* $\simeq$ Programme (accès, modification)
- *Accès concurrents* $\longrightarrow$ Problèmes

Transactions : problème 1 : (Non) Atomicité

- *Virement de compte à compte* : {C1 = C1-m; C2 = C2 + m}
  1)
  2)
  3)
  4)
  5)
  6)
- Si arrêt du programme avant *Ecrire(y)* : Incohérence
- *Atomicité* : *"Tout ou rien"*
- *"Tout"* $\Longrightarrow$ *Durabilité* (Permanence des modifications)
- *"Rien"* $\Longrightarrow$ Annulation des modifications

Transactions : problème 2 : (Non) Isolation

- *Transaction T1* : Débiter X de N
- *Transaction T2* : Créditer X de M

T1	*T2*
$x \longleftarrow Lire(X)$	$y \longleftarrow Lire(X)$
$x \longleftarrow x - N$	$y \longleftarrow y + M$
$X \longleftarrow Ecrire(x)$	$X \longleftarrow Ecrire(y)$

Problème 2 : (Non) Isolation

- X = 100; N = 10; M = 50

Exécution avec entrelacement	X_{base}	x_{T1}	y_{T2}
T1 :	100		
T1 :			
T2 :			
T2 :			
T1 :			
T2 :			

- Perte de mise à jour : X = X-N
- $\Longrightarrow$ *Isolation* : pas d'interférence pendant l'exécution

Problème 3 : (Non) Cohérence

- *Contrainte* : A = B

T1 : $\{A = A + 1; B = B + 1\}$	*T2* : $\{A = A \times 2; B = B \times 2\}$
t11 : $A_{T1} \longleftarrow Lire(A)$	t21 : $A_{T2} \longleftarrow Lire(A)$
t12 : $A_{T1} \longleftarrow A_{T1} + 1$	t22 : $A_{T2} \longleftarrow A_{T2} \times 2$
t13 : $A \longleftarrow Ecrire(A_{T1})$	t23 : $A \longleftarrow Ecrire(A_{T2})$
t14 : $B_{T1} \longleftarrow Lire(B)$	t24 : $B_{T2} \longleftarrow Lire(B)$
t15 : $B_{T1} \longleftarrow B_{T1} + 1$	t25 : $B_{T2} \longleftarrow B_{T2} \times 2$
t16 : $B \longleftarrow Ecrire(B_{T1})$	t26 : $B \longleftarrow Ecrire(B_{T2})$

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

Problème 4 : (Non) Permanence des modifications

a) *(Non) Répétabilité des lectures*

Transaction T1	Transaction T2
$a \longleftarrow Lire(A)$	
	$b \longleftarrow Lire(A)$
	Imprimer(b)
$a \longleftarrow a + 100$	
$A \longleftarrow Ecrire(a)$	
	$b \longleftarrow Lire(A)$
	Imprimer(b)

- Impressions de T2 : valeurs différentes

Problème 4 : (Non) Permanence des modifications

b) *Tuples "fantômes"*

Transaction T1	Transaction T2
	1. $V \longleftarrow Lire(A = \{a < 4000\})$
	2. *Imprimer(V)*
3. Supprimer tout a tel que $a < 4000$	
	4. $V \longleftarrow Lire(A = \{a < 4000\})$
	5. *Imprimer(V)*

- Deuxième impression de V vide

Transactions et niveaux d'isolation : SQL 92

- *Niveau 0* : lectures "sales" permises
  - D en cours de modification par T
  - Transactions autres que T, avant validation par T :
    - bloquées en modification
    - autorisées à lire D
- *Niveau 1* : lectures "sales" interdites
- *Niveau 2* : non répétabilité des lectures avant validation
- *Niveau 3* (Par défaut) : empêcher les tuples fantômes

Transactions et reprise

- Nécessité de contrôler les accès concurrents
- *Causes d'échec des transactions* :
  - Erreur logiciel ou matériel
  - Avortement par le sous-système de gestion de la concurrence
  - "Catastrophe naturelle", etc.
- Nécessité de rétablir la cohérence : *Reprise*
  - Revenir à l'état cohérent le plus proche de l'instant de l'incident
  - Mécanisme : *Journal ou Log* ("Histoire des transactions")

Schéma d'exécution des transactions

- Transactions de mise à jour : séquences
  1. Lecture (Base $\longrightarrow$ Mémoire centrale)
  2. Traitement (en mémoire centrale)
  3. Écriture (Mémoire centrale $\longrightarrow$ Base)
- *Politique "du tout ou rien"* (**A**tomicité) :

  $Action_1$
  ...
  $Action_n$
  Si toutes les actions se sont bien passées
  Alors Valider la transaction (*COMMIT*)
  Sinon Annuler ses effets (*ROLLBACK*)
  Finsi

Schéma d'exécution des transactions

- *Validation* : rendre effectives les modifications
- *Annulation* : défaire les actions de la transaction
- *Diagramme d'états d'une transaction*

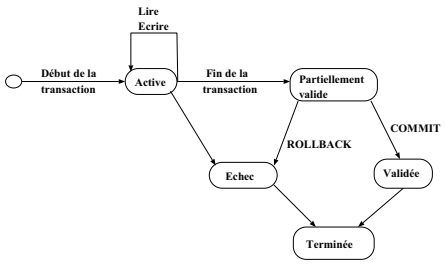


Schéma d'exécution des transactions

- *Validité partielle* :
  - Des techniques testent l'absence d'interférences
  - Des protocoles de reprise s'assurent qu'ils peuvent effectivement enregistrer les modifications
- Parfois : Annulation/Ré-exécution (UNDO/REDO) d'une opération

Journalisation, points de validation et de contrôle

- *Journal* :
  - Trace des opérations effectuées sur la base
  - Support non volatile + Archivage périodique
- *Types d'informations du journal*, pour chaque transaction :
  - <début_transaction, identification de T> ;
  - <écriture, identification de T, donnée concernée, ancienne valeur, nouvelle valeur> ;
  - <lecture, identification de T, donnée concernée>[a] ;
  - <COMMIT, identification de T>.

[a] Certains systèmes ne conservent pas de trace des opérations de lecture.

Journalisation, points de validation et de contrôle

- *Point de validation* :
  - Instant de journalisation de <COMMIT, identification de T>
  - Les actions de T ont été réussi
  - + Écritures dans le journal réussies
  - *Au-delà de ce point* : T est valide et ses MaJ sont permanentes
  - *En cas d'incident* : le système sait quoi
    - annuler ("histoire en arrière")
    - relancer, éventuellement ("histoire en avant")

Journalisation, points de validation et de contrôle

- *Points de contrôle*
  - Inscrits périodiquement dans le *Log*
  - *Période* : intervalle de temps, nombre donné de mise à jour ou combinaison des deux
  - *Prise de point de contrôle* :
    1.
    2.
    3.
    4.

  $\Longrightarrow$ les actions des transactions dont le point de validation dans le journal précède un point de contrôle pourront ne pas être ré-exécutées en cas d'incident

Points de validation et Points de contrôle

Points de validation et Points de contrôle

Transactions : Sérialisabilité et plan d'exécution

- Gestion de la concurrence fondée sur la *sérialisabilité* :

  L'exécution concurrente de n transactions doit être équivalente à (avoir le même effet pour chaque transaction que) leur exécution séquentielle

- *Propriétés (ACID)* à assurer par un gérant de transactions :
  1. *Atomicité* : politique du "tout ou rien" ;
  2. *Cohérence* : satisfaction des contraintes d'intégrité ;
  3. *Isolation* : pas d'interférence entre les transactions ;
  4. *Durabilité* : permanence des modifications effectuées.

Sérialisabilité et plan d'exécution

- A et I : par la méthode de reprise
- C : par le programmeur et le sous-système d'intégrité
- D : par les mécanismes de reprise et de contrôle de la concurrence
- *Sérialisabilité* : par le contrôle de la concurrence
- *Plan d'exécution* $\mathcal{P}$ (ou *schedule*) de n transactions :

  Ordre sur les opérations des transactions tel que pour toute transaction T_k de $\mathcal{P}$, si une opération O_i précède une opération O_j dans T_k alors O_i précède aussi O_j dans $\mathcal{P}$

- Possibilité d'entrelacement des transactions $\Longrightarrow$ $\mathcal{P}$ non unique $\Longrightarrow$ n'autoriser que les plans d'exécution corrects

Sérialisabilité et plan d'exécution

- Exécution séquentielle = *Succession* = Plan correct

  Un *plan d'exécution* $\mathcal{P}$ des transactions $T_1, \cdots, T_n$ est une *succession* (ou *serial schedule*) s'il existe une permutation Π de (1, …, n) telle que P $= < T_{\Pi(1)}; \cdots; T_{\Pi(n)} >$

- *Exécution sérialisable* : Plan correct

  Une *exécution* de $T_1, \cdots, T_n$ est *sérialisable* si et seulement si elle donne, pour chaque T_i, le même résultat qu'une succession

- *Problème* : n'autoriser que les exécutions sérialisables
  - Protocoles pessimistes : verrouillage, estampillage
  - Protocoles optimistes : contrôle à la fin d'une transaction

Contrôle de la concurrence : Verrouillage

- *Verrou* : Variable associée à un *granule* et dont la valeur indique le type d'opération possible sur un granule
- *Granule* : unité de verrouillage
  - Base, Relation, Partie de relation, etc.
- *Petite taille* :
  (+) ↗ degré concurrence
  (-) ↗ temps de son contrôle
- *Grande taille* :
  (-) ↗ temps d'attente

Contrôle de la concurrence : Types de verrous

1. *Verrou binaire* :
   - Deux états : Libre | Occupé
   - (-) Une seule transaction détient un granule
2. *Verrou partagé (Share) et verrou exclusif (Exclusive)* :
   - Accès multiples en lecture
   - En mise à jour : Un accès [+ des attentes]
   - Compatibilité des verrous :

	Partagé	*Exclusif*
Partagé		
Exclusif		

Contrôle de la concurrence : Types de verrous

3. *Verrou d'intention* :
   - Transaction demande un verrou de mise à jour ≃ verrou de lecture avec *intention* d'écriture
   - Compatibilité :

	Partagé	*Exclusif*	*Mise à jour*
Partagé	Oui	Non	Non
Exclusif	Non	Non	Non
Mise à jour	Oui	Non	Non

Gestion des transactions : Verrouillage à deux phases

- *Two Phase Locking Protocol* : le plus souvent implanté
  1. *Phase 1* (expansion) : acquisition
  2. *Phase 2* (rétrécissement) : libération et plus aucune autre acquisition
- Garantie de la sérialisabilité
- Mais verrouillage $\Longrightarrow$ Risques :
  1. Interblocage *(Deadlock)*
  2. Famine *(Livelock)*
- **Note :** *Two Phase Locking strict*

Verrouillage et Interblocage

- Attentes mutuelles
- *Prévention* : Imposer à toute transaction de verrouiller en avance tous les éléments dont elle a besoin et n'apposer aucun verrou si un de ces éléments n'est pas libre
- *Détection* : Cycle dans un graphe d'attente
  - *Nœuds* : Transactions
  - *Arc* $T1 \longrightarrow T2$: T1 *attend* un granule détenu par T2
  - Présence de cycle :
    1. *Tuer* une transaction
    2. Défaire ses actions
    3. Libérer ses ressources

Verrouillage, Interblocage et Famine

- *Exemple d'interblocage* :
- *Famine* : Attente infinie
  - *Exemple* : Priorité toujours aux mêmes transactions
  - *Solutions* : Priorités dynamiques ou *FIFO*

Contrôle de la concurrence : Estampillage

- *Estampille* : Identifiant de transaction (date de début)
- Technique fondée sur un ordre sur les *estampilles* : sérialisabilité sans interblocage
  - Plan d'exécution sérialisable s'il se conforme à l'ordre
  - *Succession* équivalente = transactions ordonnées sur les estampilles
- Granule G "marqué" par la date t de la dernière transaction qui y a accédé
- Actions sur G estampillées t', $t' \geq t$: autorisées
- Actions estampillées t'', $t'' < t$: violation de la sérialisabilité $\Longrightarrow$ Rollback

Contrôle de la concurrence : Techniques "optimistes"

- Pas de contrôle pendant l'exécution des transactions
- Mises à jour locales à la transaction
- Fin de transactions : test de non violation de la sérialisabilité
- Adaptées aux transactions ayant peu d'interférence

Reprise en cas d'incident (*Recovery*)

- Reconstruire un état cohérent à partir "du passé" (*Log*)
- État le plus proche possible de l'instant de l'incident
- Stratégie de reprise dépend de la gravité de l'incident

1. Reprise "à foid : si dégâts importants
   (a) Charger une sauvegarde de la base
   (b) Re-exécuter les transactions valides des journaux
2. Reprise "à chaud" : Défaire [et refaire] des actions

Reprise “à chaud”, Technique 1 : Mise à jour différée

- Mise à jour effective après le point de validation de la transaction
- $\simeq$ Modifications de données *“au brouillon”* (copies)
- *Cas échec* : base inchangée
- *Cas succés* : Substitution Copie/“Original” (*Shadow paging*)
- *Cas incident au moment de la substitution* : Refaire certaines actions de transactions validées

Reprise “à chaud”, Technique 1 : Mise à jour différée

Reprise “à chaud” : Technique 2 : Mise à jour immédiate

- Journalisation des modifications avant écriture dans la base avant le point de validation
- *Write Ahead Log Protocol* :
  - Implanté dans la plupart des SGBD
  - Permet la reprise même en cas d’incident entre l’écriture dans le journal et l’écriture dans la base

Write Ahead Log Protocol

WAL et reprises

- *Cas reprise à chaud* : Annuler les transactions non validées (cf. Log)
- *Cas reprise à froid* :
  1. Charger une base cohérente (exemple : B1)
  2. Charger les journaux adéquats (exemple : J2, J3)
  3. Ré-exécuter automatiquement les transactions validées
- *Remarque* : cascade de ROLLBACK
  - Rollback T1
  - T2 a utilisé des valeurs modifiées par T1

Gestion des transactions : un peu de Théorie

- Transaction (*A.C.I.D.*)
- Validation (*COMMIT*)
- Annulation (*ROLLBACK*)
- Journalisation (*LOG*)
- *Sérialisabilité* : Exécution concurrente de n transactions “équivalente” à une exécution séquentielle

Gestion des Transactions : un peu de Théorie

1. Transaction : Définitions et problèmes
2. Notion de sérialisabilité
3. Méthodes de contrôle de la concurrence

1- Transactions : Définitions

- Transaction : Séquence indivisible d'actions
- Fin d'une transaction par :
  - Succès/Validation : Effets de ses opérations sur les objets deviennent définitifs ;
  - Échec/Abandon : Annulation des effets sur les objets.
- Événements pour un système transactionnel :
  - Opérations (lire, écrire);
  - Valider/Annuler.

1- Transactions : Définitions

- Transaction $T_i = (E_i, <_i)$
- E_i = {Événements associés}
- $<_i$:
  - Ordre *partiel* sur les E_i
  - Ordre *total* sur les opérations de E_i qui concernent un même objet

1- Transactions : Définitions

- *Exécution concurrente ou histoire d'un ensemble $\mathcal{T}$ de transactions* : $T_1, \cdots, T_n = (E, <)$
  1. $E = \cup_{(1 \leq i \leq n)}(E_i)$;
  2. $<$: Ordre partiel sur E ;
  3. L'ordre $<_i$ est conservé dans $<$:
  4. Les opérations sur un même objet sont totalement ordonnées :

2- Notion de Sérialisabilité

- Sérialisabilité : Équivalence exécution concurrente, exécution séquentielle
- Définitions formelles $\longrightarrow$ Techniques et méthodes
- Plan :
  1. Exécution sérialisable
  2. Exécution sérialisable commutable
  3. Exécution sérialisable stricte
  4. Conflits et graphe de dépendances
  5. Modes de mise à jour

2.1- Exécution sérialisable

- *Exécution sérialisable* :

  Une exécution concurrente de transactions validées est sérialisable si et seulement si il existe une exécution *en série équivalente*
- *Exécution en série* (serial schedule) : Si pour tout couple (T_i, T_j) de transactions, tous les événements de l'une précèdent ceux de l'autre dans l'ordre $<$.
- *Exécutions équivalentes* : Deux exécutions d'un ensemble T de transactions sont équivalentes si et seulement si,:
  1. elles sont constituées des mêmes événements ;
  2. elles produisent le même état final des objets et les mêmes résultats pour les transactions.

2.1- Exécution sérialisable : Exemple

2.1- Exécution sérialisable : Exemple

2.2- Exécution sérialisable commutable

- Cas particulier d'exécution sérialisable
- Deux opérations O_1, O_2 sur un objet x commutent si :

  $\forall x_0$, état initial de x, $\forall T_i \ \forall T_j$

  $\{O_1^{Ti}(x); O_2^{Tj}(x)\}$ a le même effet que $\{O_2^{Tj}(x); O_1^{Ti}(x)\}$
- *Plus formellement* :
  - $Post(x, x_0, Op)$: effet de Op sur x, étant donné x_0 ;
  - $Post(T, x_0, Op)$:
    - Effet sur T de l'exécution de Op sur x_0
    - Paramètres/Valeurs retournés par Op à T

2.2- Exécution sérialisable commutable

- La *Composition* :
  -
  -
- (Op_1, Op_2) *sont commutatifs si* :
  1.
  2.
  3.

2.2- Exécution sérialisable commutable

- Notation : Commute(Op_1, Op_2)
- Remarques :
  1. Commute(lire(x), lire(x)) : toujours
  2. Commute(écrire(x_1), écrire(x_2)) si $x_1 = x_2$
- *Exécution de transactions validées commutable* si et seulement si :
  $\forall x \ Op_1^{Ti}(x) < Op_2^{Tj}(x)$
  $\longrightarrow (Valider(T_i < Op_2^{Tj}(x)) \vee Commute(Op_1, Op_2)$
- *Exemple* :

B.2.2- Exécution sérialisable commutable (fin)

Une exécution commutable est sérialisable : il existe toujours une exécution en série équivalente à l'exécution commutable (c'est l'exécution en série des transactions dans l'ordre de leurs validations).

2.3- Exécution sérialisable stricte

- Cas particulier d'exécution commutable ;
- Opérations limitées à *lire, écrire* ;
- Sans exploitation des paramètres, seules *(lire, lire)* commutent ;
- De la définition d'une exécution commutable, on a : $\forall x\ \forall T_i,\ \forall T_j$
  –
  –
  –

2.3- Exécution sérialisable stricte (fin)

- Exécution stricte caractérisée par :
  1. Dès que T_i lit un objet, d'autres transactions peuvent le lire mais pas l'écrire avant la fin de T_i ;
  2. Dès que T_i écrit un objet, aucune transaction ne peut le lire ou l'écrire tant que T_i n'est pas terminée.
- Remarque :
  – Généralisable à d'autres opérations de consultation ($\simeq$ *lire*) et de modification ($\simeq$ *écrire*) ;
  – Exploitation de la seule la commutativité des opérations de consultation.

2- Notion de Sérialisabilité : Plan

- Plan :
  1. Exécution sérialisable
  2. Exécution sérialisable commutable
  3. Exécution sérialisable stricte
  4. Conflits et graphe de dépendances
  5. Modes de mise à jour

2.4- Conflits et Graphe de dépendances

- Graphe de dépendances : Approche théorique permettant de valider les méthodes pratiques de contrôle de la concurrence
- T_i et T_j sont en *conflit* s'il existe x accédé par $Op_1^{Ti}(x)$ et $Op_2^{Tj}(x)$ et que $\neg Commute(Op_1,\ Op_2)$.
- *Graphe de dépendances* :
  $T_i \longrightarrow T_j$ (T_j dépend de T_i) si et seulement si un conflit existe entre T_i et T_j et si $Op_1^{Ti}(x) < Op_2^{Tj}(x)$
- *Sérialisabilité et Graphe de dépendances* :
  Si le graphe de dépendances ne comporte pas de circuit, alors l'exécution est sérialisable.

2.4- Conflits et Graphe de dépendances : Exemple

2.5- Modes de mise à jour et journalisation

- Annulation dépend du mode de mise à jour

1. *Mise à jour différée* :
   - Modification sur une copie des objets ("brouillon")
   - Validation par recopie du "brouillon"
2. *Mise à jour immédiate* :
   - Modification "directement" sur les objets
   - Effets peuvent être visibles avant la fin de la transaction (*dirty reads*)

2.5- Modes de mise à jour et journalisation

- cf. *Write Ahead Log Protocol*

Gestion des Transactions : un peu de Théorie

1. Transaction : Définitions et problèmes
2. Notion de sérialisabilité
3. Méthodes de contrôle de la concurrence

3- Méthodes de contrôle de la concurrence

1. *Pessismiste* : Contrôle continu
   - Détection des conflits au fur et à mesure de leur apparition
   - Si conflit : Abandon ou mise en attente de transactions
2. *Optimiste* : Contrôle par *certification*
   - À la fin de la transaction : Violation de la sérialisabilité ?
   - Adapté pour des transactions ayant peu d'interférences
   - Mal adapté avec mises à jour immédiates : des transactions peuvent utiliser des objets modifiés par une transaction qui sera annulée ($\Longrightarrow$ Risque de cascades d'abandons)

3- Méthodes pessimistes de contrôle de la concurrence

1. Estampillage : Ordre sur des dates
2. Verrouillage : Ordre par attente

3.1- Verrouillage et contrôle de la concurrence

- Verrous Exclusifs/Partagés/++ : Sérialisabilité non assurée
- cf. Types de verrous et compatibilité
- cf. *Protocole de Verrouillage à deux phases (Two-Phase Locking Protocol)* er sérailisabilité
- cf. Problèmes
  - Interblocage (*Deadlock*) : Graphe d'attente
  - Famine (*Livelock*)

3.2- Estampillage (basique) et contrôle de la concurrence

- *Estampille* : Date associée à chaque transaction
- Même estampille pour une trasnsaction et chacune de ses opérations
- Objets marqués par l'estampille de la dernière transaction qui y a accédé
- Ordre de sérialisation : Ordre total sur les estampilles
- $\simeq$ Dépendance *a priori* :
  $Estampille(T_i) < Estampille(T_j) \Longrightarrow T_i \longrightarrow T_j$.
- *Mise en œuvre* : Association à chaque objet
  1. *R(x)* : Estampille de lecture
  2. *W(x)* : Estampille d'écriture

3.2.1- Estampillage et Mise à jour immédiate

a. Conflit (écrire, lire) : Lecture de x par T estampillée t

3.2.1- Estampillage et Mise à jour immédiate

b. Conflits (lire, écrire) et (écrire, écrire) :

3.2.1- Estampillage et Mise à jour immédiate : Exemple

- T1 estampillée 10; T2 estampillée 15
- Histoire d'une exécution concurrente :
  lire$_{T1}$(x); lireT_2(y); écrire$_{T1}$(y); écrire$_{T2}$(x).
- W(x) = 8; R(x) = 12
- W(y) = R(y) = 8

3.2.1- Estampillage et Mise à jour immédiate : Exemple

3.2.2- Estampillage et Mise à jour différée

- Modification de l'estampille à la fin de la transaction
- En cas de conflit (écrire, écrire) : Validation et écriture dans l'ordre des estampilles.
- Autres méthodes : Utilisation de la sémantique (typage) des opérations.

Concurrence et Reprise : Conclusion

- Sérialisabilité et ACID"-ité"
- Verrouillage à deux phases (Interblocage)
- *Write Ahead Log Protocol*
- *Shadow Paging*
- Reprise en cas d'incident
- Sécurité par duplication (*Mirroring*)
- Organisation de l'exploitation des bases (*officier de sécurité*)
- Transactions plates : même comportement sur tous les SGBD
- Transactions imbriquées : pas de modèle d'exécution universel