Bases de données réparties
Présentation Générale

Matthieu Exbrayat

Licence Pro Informatique

2008-2009

Qu'est-ce qu'une base de donnée distribuée

- Un ensemble de données structurées, réparties entre plusieurs sites, avec un sens global.
- Pourquoi répartir ?
  - Parce que ça présente un intérêt!
  - Sites distants > localiser les accès aux données
  - Forte charge > multiplier les points d'accès
- Comment répartir ?
  - « sans » contrôle au niveau des données (plateformes applications, moniteurs transactionnels)
  - « avec » contrôle à ce niveau : SGBD réparti

Objectif du cours

- Concepts des bases de données
  - Manipulation de données
  - Création de structures (tables)
  - Gestion du SGBD
- Illustration sous Oracle
  - Créer une base, des tables, des utilisateurs
- Les SGBD distribués
  - Concepts et réalité
  - Notion de transactions réparties

Première Partie

Définition et manipulation de données

Objectifs du cours

- Savoir manipuler des données
  - sqlplus
  - Rappels SELECT, INSERT, UPDATE
- Connaître les bases de l'administration de BD
  - Installation SGBD
  - Gestion BD, schéma, utilisateurs
- Savoir construire et interroger une BD distribuée
  - Snapshots et accès distant
- Le tout avec Oracle

SQL Plus

- Lancement (sous linux)
  - > sqlplus
  - demande login et password
- Utilisation
  - Saisie directe des requêtes, terminées par « ; »
- Autre possibilité
  - Interface web isqlplus

Select

- SELECT attributs FROM tables
  [WHERE criteres_sel]
  [GROUP BY attr_group [HAVING criteres_hav]]
  [ORDER BY attr_ordre [ASC/DESC]]
- attributs : [Table1.]Attr1, [Table2.]Attr2…
  ou *
- tables : Table1 [alias1], Table2 [alias2]…
- criteres_sel : selection, jointure, IN, [NOT] EXISTS…
- attr_group : [Table1.]Attr1, [Table2.]Attr2…
- criteres_hav : similaires à criteres_sel
- attr_ordre : [Table1.]Attr1, [Table2.]Attr2…

Exemples

- SELECT * FROM Etudiant;
- SELECT nom,prenom FROM Etudiant WHERE age < 20;
- SELECT nom, prenom FROM Etudiant WHERE nom LIKE ‘DU% ‘;
- SELECT nom, prenom FROM Etudiant WHERE nom LIKE ‘DUPON_ ‘;

Exemples (2)

- SELECT nom, prenom FROM Etudiant,Note WHERE valeur<8 AND Etudiant.numetud=Note.numetud;
- SELECT nom, prenom FROM Etudiant,Note,Matiere WHERE valeur<8 AND nommat LIKE ‘BD’ AND Matiere.nummat=Note.nummat AND Etudiant.numetud=Note.numetud;

Exemple (3)

- SELECT nom,prenom,AVG(valeur)
  FROM Etudiant, Note
  WHERE
  Etudiant.numetud=Note.numetud
  GROUP BY nom,prenom
  ORDER BY nom,prenom

Création de table

- CREATE TABLE *nomtable* (
  col type [DEFAULT *val*][*contrainte col*],
  …
  [*contrainte table*]
  );
- contrainte
  - CONSTRAINT *nom type*
- contrainte col
  - UNIQUE, PRIMARY KEY, NOT NULL
  - REFERENCES *table* [(*col*)] [ON DELETE CASCADE]
  - CHECK (condition)
- contrainte table
  - UNIQUE(*col1*, *col2*,…), PRIMARY KEY(*col1*, *col2*,…)
  - FOREIGN KEY (col1, col2, ...) REFERENCES table [(*col1*, *col2*, …)] [ON DELETE CASCADE]
  - CHECK (condition)

Création de table à partir de données existantes

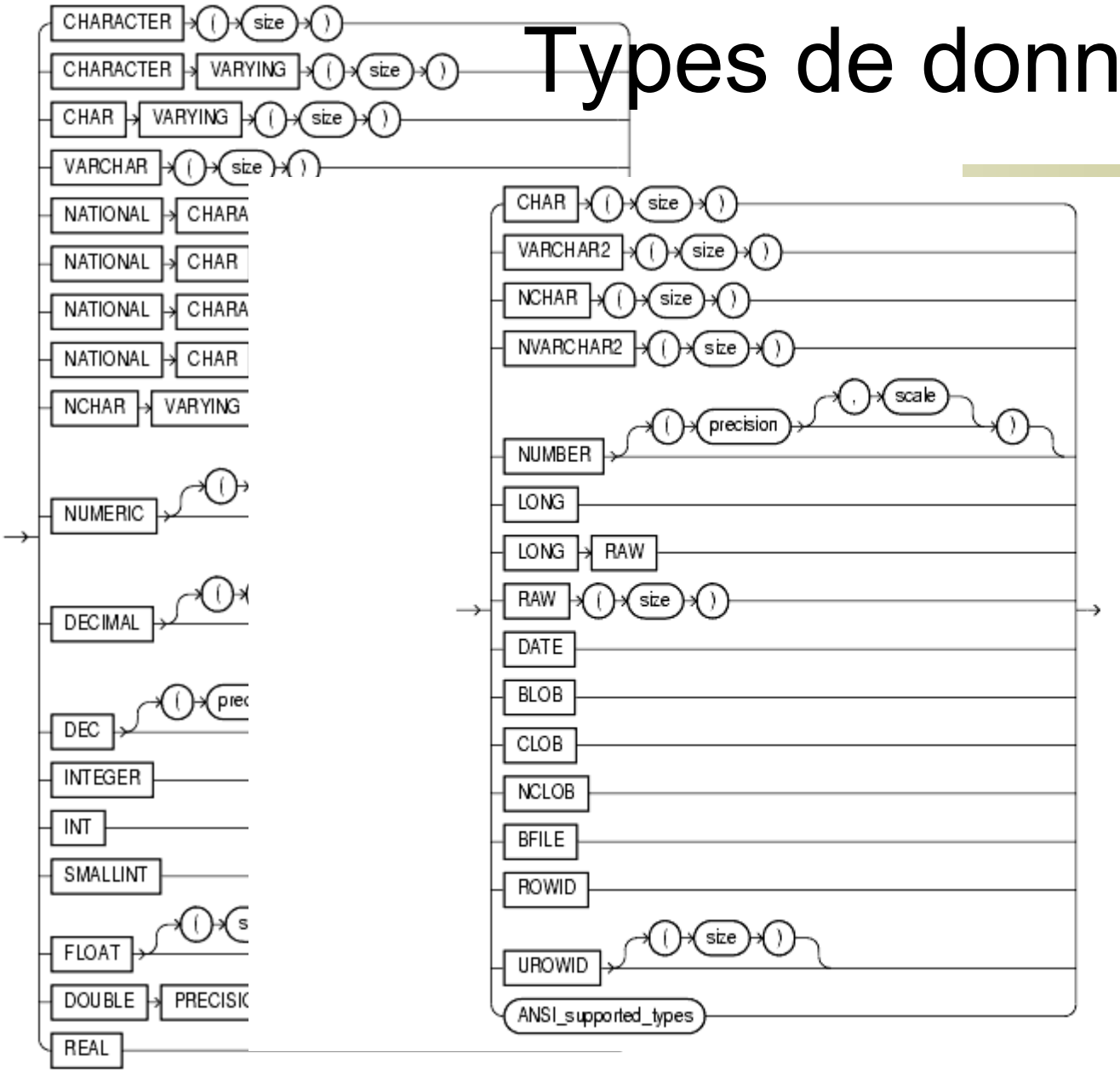
- CREATE TABLE nom (

  attr1 type 1,

  …)

  AS SELECT …

Types de données



Conversions ANSI / Oracle

ANSI SQL Datatype	Oracle Datatype
CHARACTER(n)	
CHAR(n)	CHAR(n)
CHARACTER VARYING(n)	
CHAR VARYING(n)	VARCHAR(n)
NATIONAL CHARACTER(n)	
NATIONAL CHAR(n)	
NCHAR(n)	NCHAR(n)
NATIONAL CHARACTER VARYING(n)	
NATIONAL CHAR VARYING(n)	
NCHAR VARYING(n)	NVARCHAR2(n)

ANSI SQL Datatype	Oracle Datatype
NUMERIC(p,s)	
DECIMAL(p,s)	NUMBER(p,s)
INTEGER	
INT	
SMALLINT	NUMBER(38)
FLOAT(b)	
DOUBLE PRECISION	
REAL	NUMBER

Exemples de création de table

```
CREATE TABLE Matiere (
    nummat NUMBER(5) CONSTRAINT PKMatiere PRIMARY KEY,
    nommat VARCHAR2(40) );
CREATE TABLE Etudiant (
    numetud NUMBER(5) CONSTRAINT PKEtudiant PRIMARY KEY,
    nom VARCHAR2(30),
    prenom VARCHAR2(30),
    age NUMBER(2));
CREATE TABLE Note (
    nummat NUMBER(5) CONSTRAINT REFMAT
        REFERENCES Matiere(nummat),
    numetud NUMBER(2) CONSTRAINT REFETUD
        REFERENCES Etudiant(numetud),
```

Exemple (2)

```
CREATE TABLE EMP (
    MATR NUMBER(5) CONSTRAINT PK PRIMARY KEY,
    NOME VARCHAR(2) CONSTRAINT CU UNIQUE,
    DEPT NUMBER(2) CONSTRAINT RD REFERENCES DEPT(DEPT)
        CONSTRAINT VAL CHECK (DEPT IN (1, 2, 3, 10)),
    …
    )
```

Modification et suppression de table

- ALTER TABLE nom

  [ADD (col1 type1, …)]

  [MODIFY (col1 type1, …)]

  [DROP (co1, …)]

  [DROP CONSTRAINT constr]

- ALTER TABLE nom

  RENAME TO *nouveau_nom*

- DROP TABLE

Index

- CREATE [UNIQUE] INDEX *nom_index*
  ON *nom_table* (*col1*, …)
- DROP INDEX *nom_index* [ON *nom_table*]

Création de tuple

- INSERT INTO nomtable [(col1, …)]
  VALUES (val1, …)
- INSERT INTO nomtable […]
  SELECT ...

Modification et suppression de tuple

- UPDATE table

  SET col1=exp1, col2=exp2,…

  WHERE …

- UPDATE table

  SET (col1,col2…) = (SELECT …)

  WHERE …

Notion de vue

- Une vue = une requête mémorisée
- Création : CREATE VIEW nom_vue [(structure)] AS SELECT …
- Suppression : DROP VIEW nom_vue
- Intérêt : fournit des données « déduites », donc :
  - permet le mémorisation de « bouts de requêtes » fréquents
  - permet de faire des interrogations impossibles dans

Exemple

- CREATE VIEW MOY (nom,prenom,moyenn)
  AS
  SELECT nom,prenom,AVG(valeur) FROM
  Etudiant, Note
  WHERE Etudiant.numetud=Note.numetud
  GROUP BY nom,prenom
- SELECT nom,prenom,moyenne FROM MOY
  WHERE moyenne=(SELECT MAX(moyenne)
  FROM MOY);

Notion de séquence

- Clé=valeur unique
- Attribution automatique ?
- CREATE SEQUENCE *NomSeq* [START WITH *Valeur*]
- NomSeq.NEXTVAL
- NomSeq.CURRVAL
- Utilisation ?

Deuxième Partie

Bases d'Administration

Mise en place d'une BD ?

- Installation SGBD
- Création de la Base de Données
- Création d'utilisateurs
- Création du (des) schéma(s)

Installation d'un SGBD

- Utilisateur spécifique ?
  - propriétaire du SGBD, avec un ou des groupes spécifiques
- Quels composants sont nécessaires ?
  - Serveur
  - Interface d'administration
  - Interface d'interrogation
  - Utilitaires de sauvegarde
  - Interface réseau (tns listener sous Oracle)

Fichiers de données

- Principe : le SGBD gère (toutes) ses informations sous forme de données (relationnelles)
  - Données utilisateurs
    - Données, index…
  - Données systèmes
    - Liste des tables, audit, utilisateurs…
- Ces données sont placées dans des fichiers
  - Le SGBD se charge de la gestion interne de ces fichiers

Tablespaces

- Un tablespace est une structure logique de stockage de données
- Principaux tablespaces:
  - USERS : Données utilisateurs
  - TEMP : Données temporaires (calculs)
  - SYSTEM : dictionnaire de données et procédures stockées
  - TOOLS : audit, etc.
  - RBS : Rollback

Tablespaces et fichiers

- Tablespace = organisation logique
- Fichier= organisation physique
- Un tablespace= 1 ou plusieurs fichiers

A l'intérieur d'un tablespace

- découpage de base : bloc de données
- découpage logique : segments
- un segment = une table (ou un cluster)
- si le segment est plein : extension
- Le mécanisme d'extension est géré par des règles (taille extension, pourcentage d'accroissement)
- Ces règles sont globales au tablespace
- Elles peuvent être surchargées lors de la

Gestion des tailles de segments

- Tablespace
  - CREATE TABLESPACE nomts
    DATAFILE '…', SIZE xxM
    DEFAULT STORAGE (INITIAL xxK
    NEXT yyK
    MINEXTENTS ii
    MAXEXTENTS jj
    PCTINCREASE zz)

- Table
  - CREATE TABLE (…..
)
    STORAGE (INITIAL xxK
    NEXT yyK
    MINEXTENTS ii
    MAXEXTENTS jj
    PCTINCREASE zz);

Création d'Utilisateur

- CREATE USER *nom*
  IDENTIFIED BY *passwd*
  DEFAULT TABLESPACE *ts1*
  TEMPORARY TABLESPACE *ts2*
  QUOTA … ON …
  [PROFILE *prof*];
- PROFILE=limitations

Rôles et privilèges

- privilèges = droits d'effectuer une action
  - exemples : création table, création index, etc.
- Rôle = ensemble prédéfini de privilèges
- Intérêt ?
- Affectation : GRANT
  - GRANT *priv/rôle* [ON *table*] TO *utilisateur*
- Suppression : REVOKE
  - REVOKE *priv/rôle* [ON *table*] FROM *utilisateur*

Rôles (généralement) nécessaires

- CONNECT
- RESOURCE
- Permettent de se connecter à la base et de gérer des données (création et suppression de tables, d'index, de clusters,de séquences, de vues...)

Bases de Données Réparties
Concepts et Techniques

Matthieu Exbrayat

LPI - Décembre 2007

Définition

- Une base de données répartie (distribuée) est une base de données logique dont les données sont distribuées sur plusieurs SGBD et visibles comme un tout.
- Les données sont échangées par messages
- Si les données sont dupliquées, on parle plutôt de BD répliquée

Principe fondamental

- « To the user, a distributed system should look exactly like a nondistributed system. »
(C. Date, Introduction to Database Systems)
- Autonomie locale
- Egalité entre sites (pas de site « central »)
- Fonctionnement continu (pas d'interruption de service)
- Localisation transparente
- Fragmentation transparente
- Indépendance à la réplication
- Exécution de requêtes distribuées
- Gestion de transactions réparties
- Indépendance vis-à-vis du matériel
- Indépendance vis-à-vis du Système d'Exploitation
- Indépendance vis-à-vis du réseau
- Indépendance vis-à-vis du SGBD

- Autonomie locale
  - La BD locale est complète et autonome (intégrité, sécurité, gestion), elle peut évoluer indépendamment des autres (upgrades…)
- Egalité entre sites
  - Un site en panne ne doit pas empêcher le fonctionnement des autres sites (mais perturbations possibles)
- Fonctionnement continu
  - Distribution permet résistance aux fautes et aux pannes (en théorie)

- Localisation transparente
  - Accès uniforme aux données quel que soit leur site de stockage
- Fragmentation transparente
  - Des données (d'une même table) éparpillées doivent être vues comme un tout
- Indépendance à la réplication
  - Les données répliquées doivent être maintenues en cohérence (délai possible)

- Requêtes distribuées
  - L'exécution d'une requête peut être répartie (automatiquement) entre plusieurs sites (si les données sont réparties)
- Transactions réparties
  - Le mécanisme de transactions peut être réparti entre plusieurs sites (si …)
- Indépendance vis-à-vis du matériel
  - Le SGBD fonctionne sur les différentes plateformes utilisées

- Indépendance vis-à-vis du SE
  - Le SGBD fonctionne sur les différents SE…
- Indépendance vis-à-vis du réseau
  - Le SGBD est accessible à travers les différents types de réseau utilisés
- Indépendance vis-à-vis du SGBD
  - La base peut être distibuée sur des SGBD hétérogènes
  - En théorie…

Quelques termes équivoques…

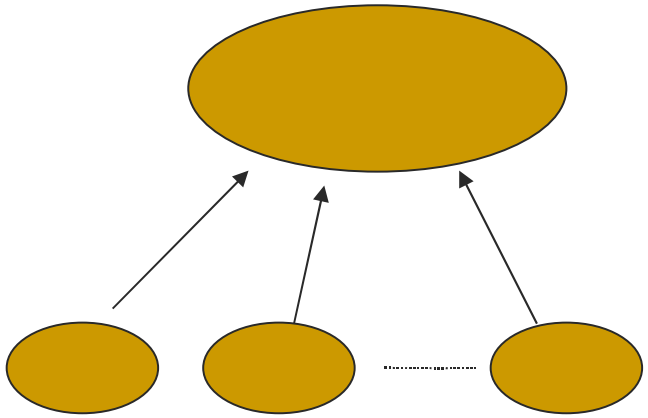
- BD distribuée
  - Un schéma global
  - Les données sont réparties sur plusieurs sites, accessibles à partir du site central ou de tous les sites
- BD fédérée
  - Chaque site a son schéma local, pas forcément inclus entièrement dans le schéma global (il y a un site central)
- Système multi-bases
  - Pas de schéma global, pas de site central. Accès à (une partie des) données distantes.
- Mais il existe d'autres définitions !
  - Ex : fédérée = approche ascendante partielle …

Conception de BD répartie

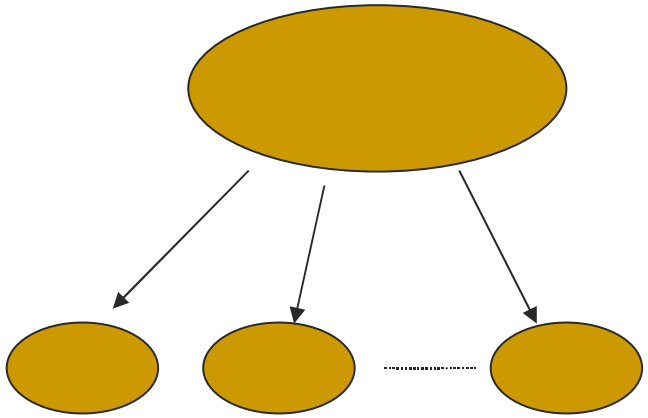
- On ne met en place une BD répartie qu'en cas de réel besoin
  - Démarche de conception délicate
  - Gestion complexe
  - L'évolution du SI peut invalider la solution retenue…
- Des raisons valables :
  - Volumes de données, sites distants, etc.
  - Fusions de SI

Deux approches de conception

- Conception ascendante
  - Part de l'existant
  - Intègre bases locales dans schéma global

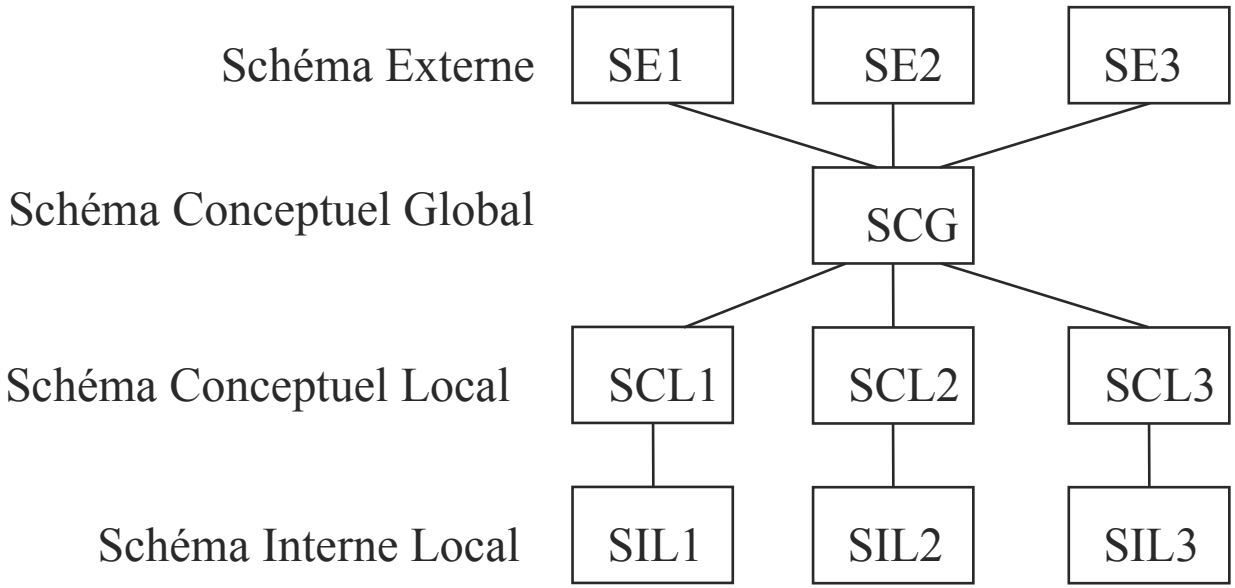


- Conception descendante
  - On part du schéma global
  - On le scinde en schémas locaux



- Conception descendante
  - On part de zéro (nouvelle base)
  - Recherche de performance (pas forcément de répartition géographique)
  - Assez peu fréquent
- Conception ascendante
  - Distribution pré-existante
  - Nécessite consolidation, uniformisation (« réconciliation sémantique »)
    - Identifier les données semblables
    - Accorder leurs types, gérer leur cohérence…
    - Interfacer ou adapter les SGBD…
  - Ex : fusion, mise en place DW

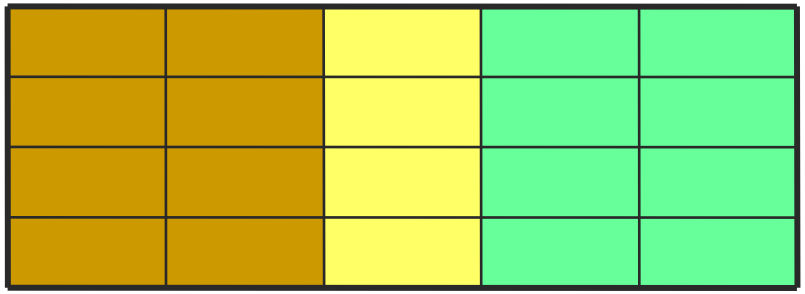
Schémas d'une BD répartie



Fragmentation des données

- Fragmentation horizontale
  - Les tuples sont répartis

- Fragmentation verticale
  - Les tuples sont découpés et fragmentés
  - Nécessite colonne commune (clé ou unique) dupliquée



Fragmentation horizontale

- En conception descendante
  - Adéquation géographique
  - Recherche de performance (I/O, traitements)
- En conception ascendante
  - Des données comparables dans différentes bases
  - Pb : consolidation correcte (unicité des clés, types des attributs…)

Fragmentation verticale

- On projette la table sur des attributs différents suivant site.
- Comme frag. horizontale, peut correspondre à consolidation ou recherche de perf.
- La reconstruction des tuples doit être possible (et validée)
- Mêmes problèmes que f.h.

Frag. horizontale dérivée

- Placer deux tables en relation de manière à localiser les jointures
- Une des deux tables doit être fragmentée en fonction de l'autre (semi jointure)
- Plutôt en approche descendante
- Peut introduire des redondances

Mise en pratique de la fragmentation

- Historique : des SGBDs distribués
  - R*, etc.
  - Plutôt expérimental
- Dans les SGBD commerciaux actuels
  - Pas de fragmentation explicite au niveau du schéma
  - Assemblage = création de vue (ou de snapshot)
  - Distribution des données ?
    - Une solution = triggers

Mise en œuvre sous SQL (Assemblage)

- Frag. Horizontale
  - CREATE VIEW V1
    AS SELECT Table1.cle, Table1.attr1
    FROM Table1@site1
    UNION
    SELECT Table2.cle, Table2.attr1
    FROM Table2@site2

Mise en œuvre sous SQL (Assemblage)

- Frag. Verticale
  - CREATE VIEW V1
    AS SELECT Table1.cle, Table1.attr1,
        Table2.attr2
    FROM Table1@site1, Table2@site2
    WHERE Table1.cle=Table2.cle
- Remarque :
  - l'attribut de fragmentation n'est pas forcément la clé primaire…
  - En frag. verticale, il faut au néanmois que ce soit une clé

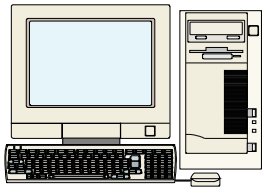
Requête répartie

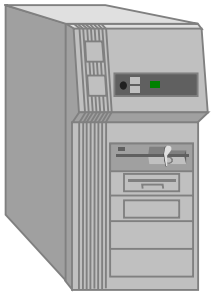
- Les opérations précédentes (par exemple) accèdent à des données situées sur différents sites.
- Le SGBD gère les accès distants en répartissant la requête
  - Exécution distante de la sous requête
  - Récupération de données résultat
  - Assemblage local (requête en lecture)

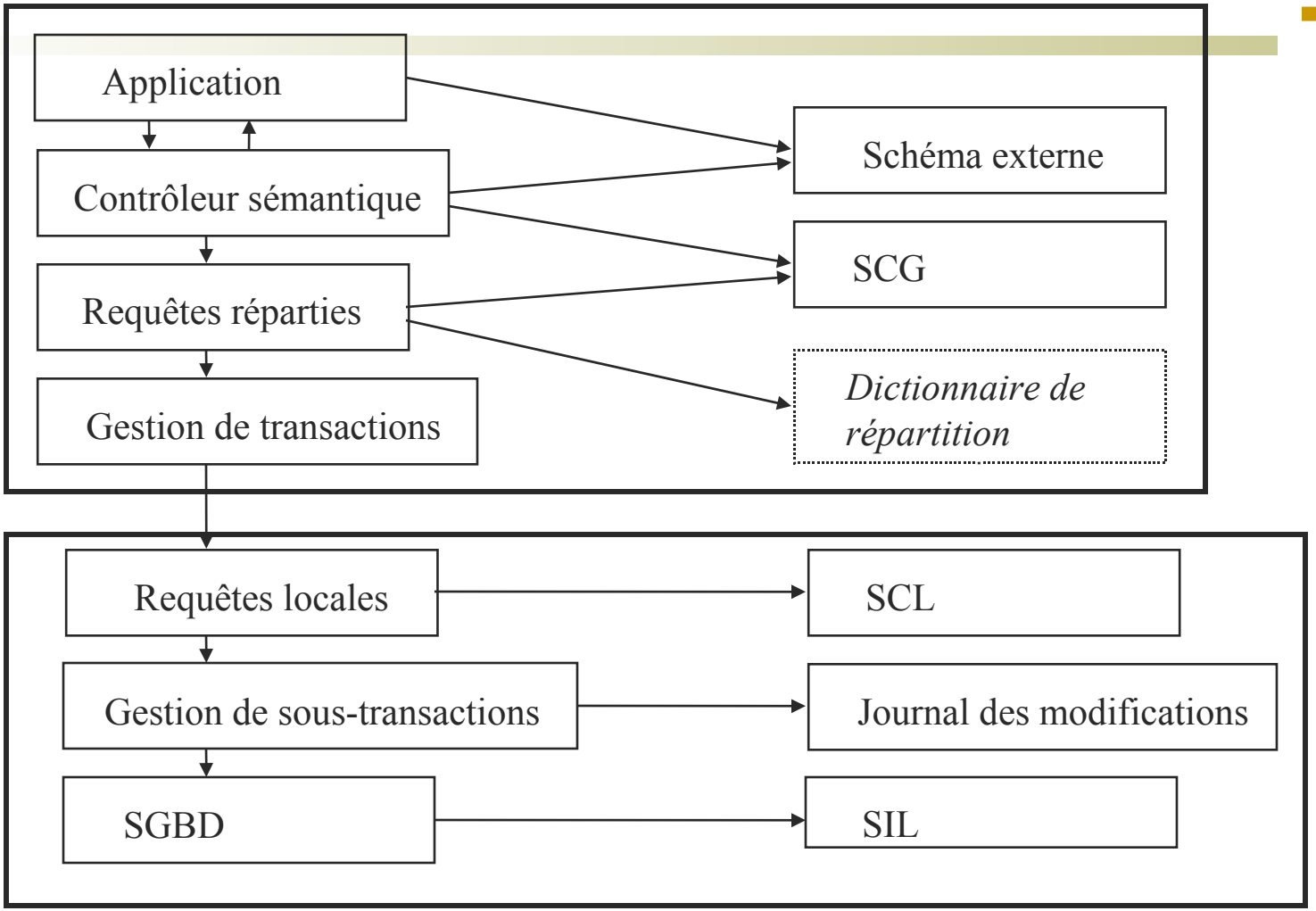
Gestion de l'hétérogénéité

- Hétérogénéité « sans problème »
  - SE et réseau : géré par SGBD (si « bon » SGBD)
  - Version de SGBD : niveau de SGBD le plus ancien
- Hétérogénéité plus délicate
  - SGBD : pb des dialectes de SQL
    - passerelles entre SGBD
    - Ex : ODBC (au départ sous Windows mais porté sous d'autres OS)
    - Ex : passerelles propriétaires SGBD à SGBD

BDR comment ça marche







Communication Inter-sites

- Chaque SGBD dispose d'un démon permettant les connexions distantes, sur un mode client - serveur
  - *Listener (médiateur)*
- Chaque SGBD dispose d'une table des BDs accessibles
  - *Nom >> doit être unique !!!*
  - *Adresse*
  - *Protocole*
- Cette approche permet aussi un équilibrage de charge transparent…

Exemple : Oracle

- Permet la distribution et la réplication
- Assure une bonne transparence à différents niveaux

- Système de nommage simple
  - sales.france.europe.computers
- Accès BD distante : LINK
  - CREATE DATABASE LINK sales…
- Accès table distante : schéma.table@base
  - svc_maint.emp@sales.france.europe.computers
- Lien public, lien privé

Transparence

- Localisation : synonymes
  - CREATE PUBLIC SYNONYM employes
    FOR
    svc_maint.emp@sales.france.europe.computers
- Requêtes et transactions
- Opérations internes
- Réplication

Caractéristiques

- Autonomie des sites
  - Gestion indépendante, upgrades localisés…
- Sécurité
  - Les utilisateurs et leurs rôles doivent être connus sur chaque site accédé
  - Possibilité d'utiliser un Security Server
  - Encryptage
- Administration globale : enterprise manager

Mise en oeuvre en SQL (Insertion avec les triggers Oracle)

```
CREATE TRIGGER Tr1
INSTEAD OF INSERT on Table
BEGIN
IF :New.cle < 1000 THEN
    INSERT INTO Table1@site1(cle,attr1)
    VALUES(:New.cle,:New.attr);
ELSE
    INSERT INTO Table2@site2(cle,attr2)
VALUES (:New.cle,:New.attr);
END IF;
END;
```

Complément sur triggers

- Utilisables également pour update et delete (:New et :Old) pour
  - Suppression (:Old)
  - Modification hors attribut de distribution (:New)
  - Modification attribut de distribution (:New et :Old pour vérifier si bon site)
- Doit être écrit « à la main » par le dba…

Gestion des contraintes distribuées

- Types de contraintes
  - Domaine
  - Unicité
  - Intégrité référentielle
- Dans un SGBD centralisé, c'est bien géré
  - Dans la définition des tables (principalement)
- Le côté manuel de la distribution limite les possibilités de gestion des contraintes
  - Utilisation de vues >> pas de contraintes

Exemples

- Check / NOT NUL
  - Se gère par fragment
- Unicité
  - Gérable par fragment
  - Globalement ? Réplication OK, Fragmentation KO
  - Utilisation de trigger
  - Site maître pour numérotation >> perte autonomie
- Intégrité Référentielle
  - Pas géré en inter-bases
  - Réplication OK, Fragmentation KO
  - Utilisation trigger
- Ca devient vraiment lourd à gérer…

Liens BD : notions avancées

- 3 types de liens :
  - Connected user
    - Mode de base : même utilisateur en local et distant
    - Ce n'est pas nécessairement le créateur du lien
  - Current user
    - Si lien utilisé à partir de procédure, identification en temps que propriétaire de la procédure
    - Intérêt ?
  - Fixed user
    - Utilisateur fixe.
    - Intérêt ?

Exemples

- CREATE DATABASE LINK ventes.ulp.fr
  USING ‘ventes’;
- CREATE DATABASE LINK ventes.ulp.fr
  CONNECT TO scott IDENTIFIED BY tiger
  USING ‘ventes’;
- CREATE DATABASE LINK ventes.ulp.fr
  CONNECT TO CURRENT_USER
  USING ‘ventes’;
- Remarque : USING -> nom du service!

Liens BD : notions avancées

- Visibilité :
  - Public
    - Pratique si utilisé par un grand nombre d'utilisateurs
  - Private (défaut)
    - + sécurisé, accessible uniquement au propriétaire et à ses sous-programmes
  - Global
    - Accessible de n'importe quelle base, mais suppose une gestion centrale (directory server)

exemples

- CREATE PUBLIC DATABASE LINK ventes.ulp.fr
  USING ‘ventes’;
- CREATE PUBLIC DATABASE LINK ventes.ulp.fr
  CONNECT TO scott IDENTIFIED BY tiger
  USING ‘ventes’;
- CREATE PUBLIC DATABASE LINK ventes.ulp.fr
  CONNECT TO CURRENT_USER
  USING ‘ventes’;
- Différence avec privé ?

Opérations sur liens

- Fermeture
  - Utile si forte charge
  - ALTER SESSION CLOSE DATABASE LINK lien;
- Suppression
  - DROP DATABASE LINK lien

Infos utiles…

- Tables
  - DBA_DB_LINKS : tous les liens
  - ALL_DB_LINKS : tous ceux accessibles par l'utilisateur
  - USER_DB_LINKS : tous ceux qui lui appartiennent
  - V$DBLINK : tous les liens ouverts par la transaction
- Paramétrage
  - OPEN_LINKS : nombre de liens simultanés dans une même session (défaut 4, 0 = pas de limite)
  - (INIT.ORA)

Liens partagés

- Un lien peut être partagé par plusieurs utilisateurs
- Mot clé « shared »
- **CREATE SHARED DATABASE LINK nomlien**
  [CONNECT TO util IDENTIFIED BY passwd]|
  [CONNECT TO CURRENT_USER]
  AUTHENTICATED BY schema IDENTIFIED BY passwd
  [USING 'service']
- Authentification pour le lien, connexion au schéma distant suivant mode de connexion choisi
- Intérêt : si beaucoup de connexions, sinon surcoût (pooling)
- Peut être connecté vers serveur à processus dédiés ou partagés

Notion de nommage global

- Oracle peut utiliser le nommage global
  - Chaque base est identifiée par un nom et un domaine. Le nom complet doit être unique
  - Utile avec les liens, indispensable pour la réplication
- Paramétrage
  - GLOBAL_NAMES (INIT.ORA)

Manipuler les noms globaux

- SELECT NAME,VALUE FROM V $PARAMETER WHERE NAME=« db_domain »;
  - Modifier : dans INIT.ORA
- SELECT * FROM GLOBAL_NAME;
- ALTER DATABASE RENAME GLOBAL_NAME TO nom.domain;

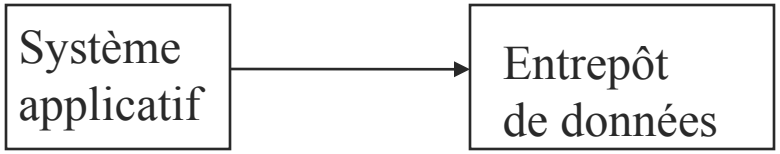
Bases de données répliquées

- Ensemble de bases de données identiques, dont une appelée *copie maître* permet de créer les autres appelées *copies esclaves*.

- Techniques de mise en cohérence
  - Synchrone: la mise à jour des copies est faite dans la même transaction
  - Asynchrone: la mise à jour des copies est faite le plus tôt possible

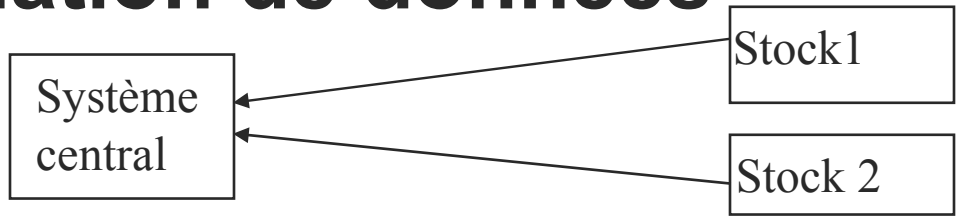
Bases de données répliquées

- **Alimentation d ’entrepôt de données**



- **Dissémination de données**

- **Consolidation de données**

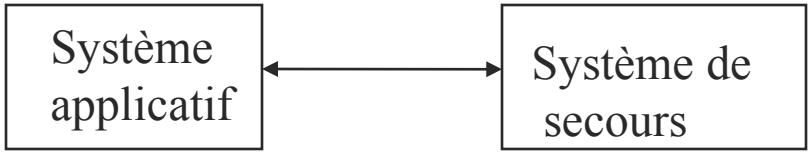


Bases de données répliquées

- **Découpage d'un processus par activité**

- **Accès délocalisé**

- **Systèmes 24h/2 4**



Bases de données répliquées

- **Réplication sans conflits**

  En évitant les mises à jour multiples (réplication asymétrique)

  - **Système maître unique**
    - **Alimentation des entrepôts de données**
    - **Dissémination d'information**
    - **Consolidation d'information**
  - **Système maître désigné en dynamique**
    - **Découpage d'un processus par activité**

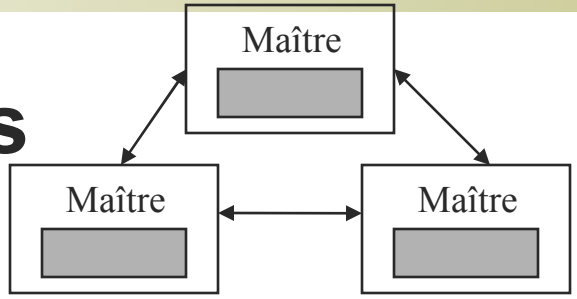
- **Réplication avec résolution des conflits**

  Une règle de priorité permet de résoudre les conflits (r. symétrique)

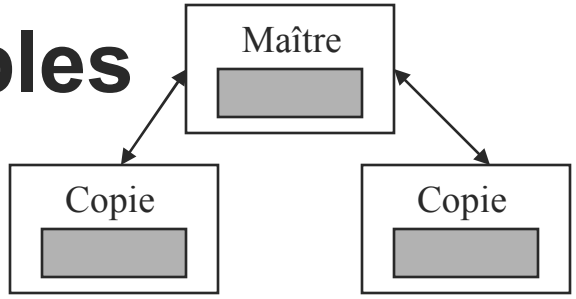
  - **Systèmes maîtres multiples**
    - **Accès délocalisé**
    - **Système 24h/24**

Bases de données répliquées

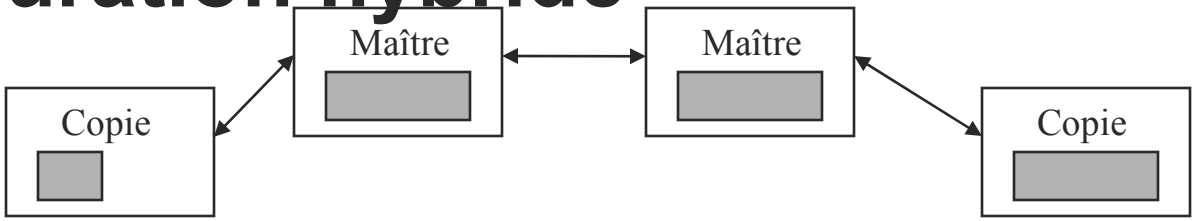
- **Copies maîtres multiples**



- **Copies esclaves modifiables**



- **Configuration hybride**



Réplication

- Réplication simple (maître-esclave, read only)
  - Snapshots
  - MAJ : complète ou incrémentale
  - Snapshots complexes : plusieurs tables, pas de MAJI
- Réplication avancée (multi maîtres, rw)
  - multimaître, hybride
  - réplication objet ou groupe
  - site et catalogue de réplication

Conflits d'accès

- Unicité, modif, suppression
  - Des méthodes dans les deux premiers cas
  - pour suppr, suggère l'utilisation d'un flag « suppr »

- Possession statique : une donnée $\in$ à un seul serveur
- Possession dynamique : les serveurs demandent l'exclusivité
- Possession partagée : les conflits éventuels sont gérés de manière asynchrone

- Gestion synchrone : synchro immédiate

Création d'une vue matérialisée

- CREATE MATERIALIZED VIEW nommv
  [TABLESPACE … STORAGE …]
  [REFRESH FAST|COMPLETE|FORCE
    START WITH sysdate
    NEXT sysdate+1 *// en jours…*
    WITH PRIMARY KEY *// si possible …*
    USING ROLLBACK SEGMENT …]
  AS
  SELECT…

Vue simple

- Rafraîchissement rapide possible
- Repose sur clé
  - WITH PRIMARY KEY
- Repose sur ROW ID
  - WITH ROW ID
- Repose sur une requête adaptée
  - sous requête de type exists
  - union

Des vues « simples »…

- CREATE MATERIALIZED VIEW info_produit REFRESH FAST AS
  SELECT * FROM info_produit ip
  WHERE id =10 AN EXISTS
  (SELECT * FROM descr_produit dp
  WHERE…)
  UNION
  …
- Cela reste rafraîchissable en fast !

Des vues plus complexes…

- Pas fast refresh si contient :
  - DISTINCT, UNIQUE
  - INTERSECT, MINUS, UNION ALL
  - CONNECT BY
  - Jointure
  - Agrégation
  - UNION si pas les mêmes types (ou requête complexe)
- Pour tester rafraîchissement rapide
  - DBMS_VIEW.EXPLAIN_MVIEW(‘vue’);

Compléments sur les V.M.

- pour que le REFRESH fonctionne il faut paramétrer Oracle en conséquence…
  - Noms globaux obligatoires
  - Utilisation d'un lien en fixed user
- Pour autoriser le fast refresh if faut créer un snapshot log sur la table source
  - CREATE SNAPSHOT LOG ON Table
    WITH PRIMARY KEY
    TABLESPACE ... STORAGE (…)

Refresh gérable par le biais de fonctions PL/SQL

- DBMS_REFRESH.REFRESH('vue');

Groupe de rafraîchissement

- DBMS_REFRESH.MAKE(
  name =>’nomgrp’,
  list=>’’,
  next_date=> SYSDATE,
  interval=> ‘SYSDATE+… ‘,
  implicit_destroy=> FALSE,
  rollback_seg => ‘’,
  push_deferred => TRUE,
  refresh_after_errors => FALSE);

- DBMS_REFRESH.ADD(
  name=>’nomgrp’,
  list=>’nomvue’,
  lax=TRUE);

Vues matérialisées modifiables

- Intérêt ?
- Ajouter « FOR UPDATE » avant le « AS SELECT… »
- Rattacher à un groupe de vues matérialisées

Groupe de vues matérialisées

- Simplification de l'administration
- D'autres avantages…
- Création du groupe :
  - DBMS_REPCAT.CREATE_MVIEW_REPGROUP(
    gname=>'nomgrp',
    master=>'base d'origine',
    propagation_mode=>'ASYNCHRONOUS');
- Ajout dans le groupe
  - DBMS_REPCAT.CREATE_MVIEW_REPOBJECT(
    gname=>'nomgrp',
    sname=>'schema',
    oname=>'nommv',
    type=>'SNAPSHOT',
    min_communication>='TRUE');

Privilèges

- ALTER ANY SNAPSHOT
- CREATE ANY SNAPSHOT
- DROP ANY SNAPSHOT
  - Plutôt pour le DBA…
- CREATE DATABASE LINK
  - Utile car permet d'utiliser les liens privés
- CREATE SNAPSHOT
  - permet ajout, modif et suppr
- CREATE VIEW
  - Permet gestion de vue sous-jacente au snapshot

Vues multi tiers

- On peut avoir des vues matérialisées qui reposent elles mêmes sur des vues matérialisées
- Contraintes : essentiellement utilisation des clés primaires.

Systèmes multi maîtres

- Intérêt :
  - Équilibrage de charge
  - Résistance aux pannes
  - Interopérabilité entre applications
- Cohérence ?
  - Synchrone : MAJ immédiate
  - Asynchrone : files d'attente (et d'erreur…)

Mise en oeuvre

- Rôles
  - Administrateur
    - Ajout de sites, mise en sommeil, etc.
  - Propagateur
    - Envoi des requêtes
  - Récepteur
    - Mise en œuvre des requêtes reçues

Structures

- Liens planifiés
  - Liens avec utilisateur fixe + planification des envois (transactions)
- Purge
- Groupe maître
  - Contient les objets répliqués
  - Il peut y en avoir plusieurs…
  - Notion de site de définition
- Site maîtres
  - Ajoutés sur le site de définition

Résolution de conflits

- Méthodes proposées par Oracle:
  - Plus récente modif (timestamp)
  - Réécriture
  - Max et min
  - Moyenne et somme
  - Estampillage
  - Priorité de groupe
  - Priorité de site

Résolution de conflit

- Plus récente modif
  - La plus récente est la bonne
  - (pas de lien entre valeurs)
  - Suppose datage uniforme
- Réécriture
  - Essentiellement pour mono maître

Résolution de conflit

- Additive
  - On ajoute la différence
  - OK pour transactions financières par ex.
- Moyenne
  - OK pour donnée évoluant vers moyenne
- Abandon
  - OK avec mono maître

Résolution de conflit

- Plus ancienne date
  - Utilisation limitée… (mono maître)
- Maximum, Minimum
  - Pour des données s'y prêtant
- Groupes prioritaires
  - Priorité suivant la valeur d'une colone donnée
- Site prioritaire

Autres types de conflits

- Insertion
  - Séquences
- Suppression
  - Colonne de suppression

Eviter les conflits

- Groupes de colonnes
  - On peut modifier sur deux groupes indépendants sans générer de conflit
- Site propriétaire
  - Granularité réglable
- Possession dynamique

Compléments sur la réplication

- Template de réplication
  - Pour simplifier réplication massive
  - Ex : copies sur machines nomades
  - Description des tables distribuées
- Réplication d'autres informations
  - Triggers
  - Types d'Objets
  - …

Gestion de Transactions dans les Systèmes Distribués

Structure d'une transaction

- " Pour faire un contrat deux ou plus de parties négocient et parviennent à un accord. L'accord est scellé en co-signant un document ou par un autre acte. Si les parties se soupçonnent ou veulent s'assurer, ils rénumèrent un intermédiaire pour coordonner la validation de la transaction"
  - *Exécution durable de tout le contrat*
- **Primitives délimitant une transaction**
  - Début_transaction
  - Fin_transaction
- **Primitives conditionnant la terminaison d'une transaction**
  - Valider_transaction (commit) *Etat final*
  - Abandonner une transaction (rollback) *Etat initial*

Exemple

TABLES VOL(VNO, DATE, A_DEP, A_ARR,
SIEGES_VENDUS,CAPACITE)
CLIENT(CNOM, ADRESSE, SOLDE_COMPTE)
VOL_CLIENT(VNO, DATE, CNOM, SPECIAL)

Début_transaction. *Réservation*

entrées (vol_no, date, nom_client);
EXEC SQL SELECT SIEGES_VENDUS, CAPACITE
INTO temp1,temp2
FROM VOL
WHERE VNO = vol_no
AND DATE = date;
si temp1 = temp2 alors
début
afficher (“ plus de places ”);
Abandonner_transaction
fin

Exemple (suite)

```
sinon début
    EXEQ SQL UPDATE VOL
        SET SIEGES_VENDUS = SIEGES_VENDUS+1
        WHERE VNO = vol_no
        AND DATE = date;
    EXEC SQL INSERT
        INTO VOL_CLIENT(VNO, DATE, CNOM, SPECIAL)
        VALUES(vol_no, date, nom_client, null);
    Valider_transaction;
    afficher( " réservation effectuée ")
    fin
    fin_si
Fin_transaction.
```

Gestion de la Concurrence

- Une transaction atomique et cohérente ne met pas en cause l'intégrité de la base de données.
- Lorsque plusieurs transactions sont exécutées en concurrence, leurs opérations peuvent interférer de sorte à aboutir à de résultats incorrects.
- *Exemples:* ***cc*** *compte courant* ***ce*** *compte d'épargne*

mise à jour perdue **lecture incorrecte**

T1	**T2**	**T3**	**T4**
①⑤ L(cc)	② L(cc)	① L(cc)	④ L(cc)
cc=cc+500	cc=cc+1000	cc=cc-500	⑤ L(ce)
③ E(cc)	④ E(cc)	② E(cc)	imprimer(cc,ce)
valider	valider	③ L(ce)	valider
		ce=ce+500	
		⑥ E(ce)	
		valider	

Propriétés d'une transaction

- **ATOMICITE**
  les opérations d'une transaction seront toutes entièrement exécutées, en cas de problème avant terminaison, les opérations exécutées seront annulées.
- **COHERENCE**
  la transaction est un programme correct qui fait passer la base de données d'un état cohérent vers un autre état cohérent.
- **ISOLATION**
  les résultats intermédiaires d'une transaction (avant terminaison) ne seront pas accessibles par les autres transactions
- **DURABILITE**
  les résultats d'une transaction sont permanents après terminaison et ne doivent être altérés par aucun type de panne

Degrés de cohérence

- T ne modifie pas les données sales des autres transactions
- T ne valide pas d ’écritures avant d ’avoir fini toutes ses écritures
- T ne lit pas les données sales des autres transactions
- Les autres transactions ne salissent pas les données lues par T jusqu ’à ce que T soit terminée.

0

1

2

3

Risques liés à une mauvaise isolation

- Lecture sale
  - Lecture d ’une donnée dont la valeur n ’a pas été validée : la donnée peut avoir une valeur différente ou ne plus exister à terme...
- Lecture floue (non répétable)
  - Lectures répétées d ’une valeur en cours de modification : les calculs sont biaisés, ou la données n ’est plus disponible...
- Fantômes
  - Des tuples sont ajoutés par T2 alors qu ’un requête a été posée sur leur table par T1: il y a des tuples « fantômes »

Niveaux d ’isolation

- Read uncommitted
  - Les trois problèmes peuvent se présenter ...
- Read committed
  - on élimine les lectures sales
- Repeatable read
  - Il ne reste que les fantômes
- Anomaly serializable
  - Tout se passe bien
  - Mais cela nécessite plus de contrôles et de verrous...

Exécutions en série
Exécutions sérialisables

- L'exécution en série des transactions consiste à ce que, pour tout couple de transactions, toutes les opérations de l'une se soient exécutées avant toute opération de l'autre *(performances)*
- Une exécution est sérialisable si elle produit les mêmes résultats et les mêmes effets sur la base que l'exécution en série des mêmes transactions. La sérialisabilité d'une exécution concurrente de transactions est le critère habituel de correction des méthodes de contrôle de concurrence.

Exécutions en série
Exécutions sérialisables

Exécution série

T1	**T2**
	L(y)
	y=y-200
	E(y)
	L(z)
	z=z+200
	E(z)
L(x)	
x=x-100	
E(x)	
L(y)	
y=y+100	
E(y)	

Exécution sérialisable

T1	**T2**
	L(y)
L(x)	
	y=y-200
x=x-100	
	E(y)
E(x)	
L(z)	
L(y)	
z=z+200	
y=y+100	
	E(z)
E(y)	

Exécution non-sérialisable

T1	**T2**
L(x)	
x=x-100	
E(x)	
	L(y)
	y=y-200
L(y)	
	E(y)
y=y+100	
E(y)	
	L(z)
	z=z+200
	E(z)

Gestion de la concurrence et bases de données réparties

- **Bases de données réparties**
  Si une base n'est pas dupliquée et si chaque ordonnancement local est sérialisable, alors leur union (ordonnancement global) est aussi sérialisable

- **Bases de données dupliquées (copies multiples)**
  Les ordonnancements capables de maintenir la cohérence des bases de données dupliquées sont appelées " one-copy-serialisable ", et respectent les conditions suivantes:
  - chaque ordonnancement local est sérialisable.
  - deux opérations conflictuelles doivent respecter le même ordre relatif dans les ordonnancements locaux où ils apparaissent.

Méthodes de gestion de concurrence

- **Approche pessimiste**
  il est considéré que plusieurs transactions seront en conflit, la synchronisation des exécutions concurrentes se fera au début des transactions

  Utilisée dans le cas de beaucoup de transactions partageant peu de données: systèmes d'information opérationnels

- **Approche optimiste**
  il est considéré que peu de transactions seront en conflit, la synchronisation est reportée à la fin des transactions

  Utilisée dans le cas de peu de transactions partageant beaucoup de données: systèmes d'aide à la conception

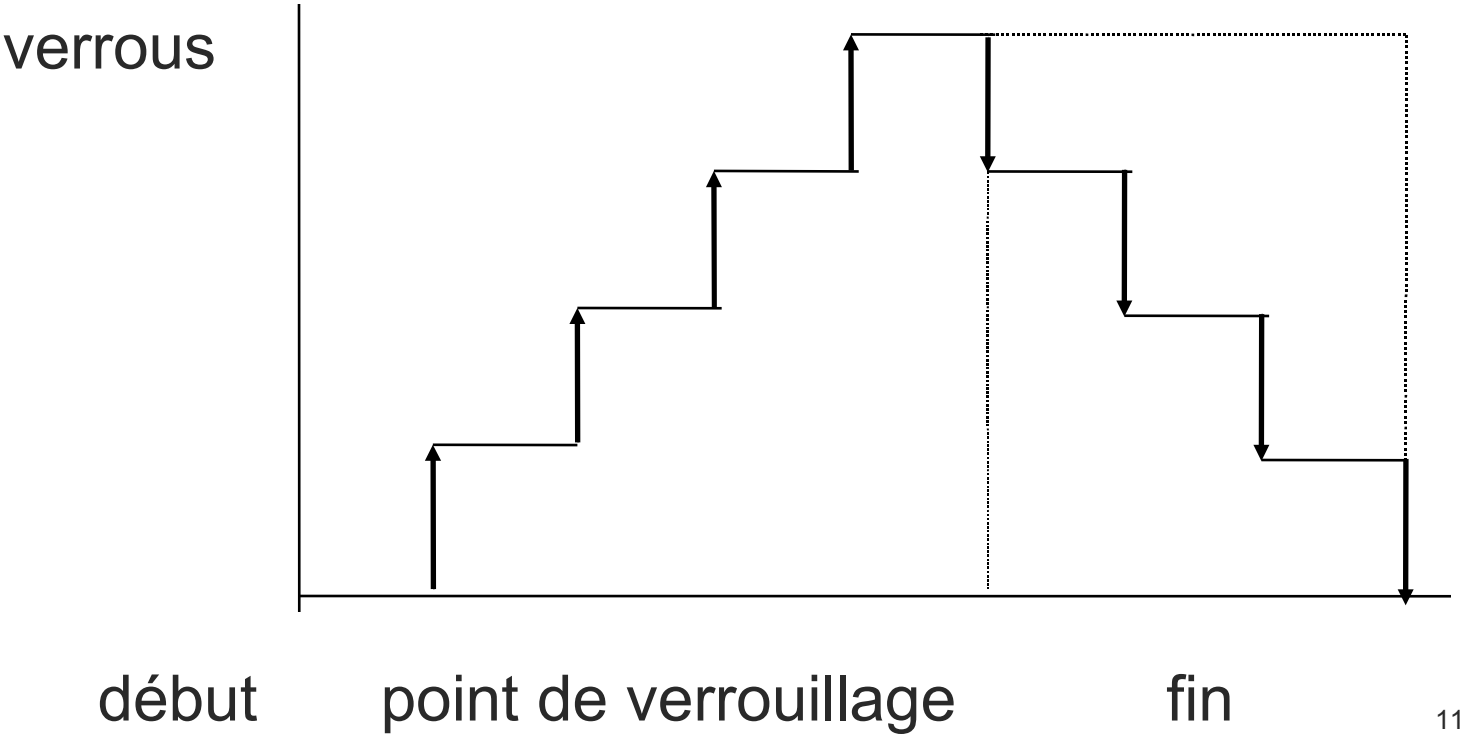
Verrouillage (Locking)

- La synchronisation des transactions est obtenue en appliquant des verrous sur un granule de la base. La taille de ces granules, appelée granularité du verrouillage, a un impact certain sur les performances. Certains systèmes mettent en œuvre des granularités différentes à la demande.
- Les verrous demandés par les transactions sont gérés dans des tables de verrouillage.
- Compatibilité des verrous

Verrou demandé	Verrou actuel : pas de verrou	partagé	exclusif
pas de verrou	oui	oui	oui
partagé	oui	oui	non
exclusif	oui	non	non

Verrouillage (Locking)

- Verrouillage à deux phases
  - phase de croissance: obtenir des verrous
  - phase de rétrécissement: libérer les verrous



Estampillage (Timestamp ordering)

- L'estampillage consiste à ordonner les transactions réparties lors du lancement de leur exécution et à imposer que les opérations d'accès aux données respectent l'ordre préétabli. Pour ce faire, un numéro d'ordre unique appelé estampille est affecté à chaque transaction et à chaque granule accédé.
- Dans un système réparti l'unicité de l'estampille est obtenu par la synchronisation des horloges (Time Service).
- **Estampillage basique**: une transaction Ti accède au granule dont l'estampille est j.
  Si j≤i alors l'accès par Ti respecte l'ordre d'arrivée des transactions et peut être exécutée.
  Sinon Ti sera abandonnée et reprise avec une nouvelle estampille
- **Estampillage conservatif**: l'ordonnanceur retardera artificiellement les opérations pour éviter les abandons
- **Estampillage multi-versions**: les mises-à-jour ne modifient pas la base mais une copie de celle-ci

Gestion d'inter blocages

- Tout mécanisme d'allocation exclusive de ressources peut aboutir à un inter blocage (deadlock)
- Un inter blocage survient quand deux (ou plus) transactions se mettent en attente sur des ressources verrouillées de manière croisée.
- Un inter blocage est un phénomène permanent; il ne disparaîtra que par une intervention externe: utilisateur, opérateur système, système d'exploitation, SGBD,...
- T1 T2

  lire(x) lire(y)

  lire(y) lire(x)
- Un outil d'analyse des inter blocages est le graphe d'attente GA, qui est un graphe orienté dont les arcs représentent une relation d'attente entre transactions. Un arc Ti→Tj indique que Ti attend que Tj libère un verrou. Les circuits du GA indiquent des inter blocages

Gestion d'inter blocages: méthodes

- PREVENIR
  Pour qu'un inter blocage soit impossible il faut éviter de mettre en exécution les transactions qui pourraient rentrer en conflit, avec une pré-déclaration des données utilisées.
- EVITER
  Ordonner les ressources et demander que les transactions respectent l'ordre d'accès. Se servir des estampilles pour affecter des priorités.
- DETECTER ET RESOUDRE
  La détection se fait par l'identification des cycles dans les graphes d'attente ou par des temporisations.
  La résolution se fait par l'abandon d'une ou plusieurs transactions " victimes ".
  Critères de choix:
  - la quantité de travail déjà effectué par les transactions
  - le coût de l'abandon en termes de mises-à-jour à défaire
  - la quantité de travail restant à effectuer
  - le nombre de cycles concernés par chaque transaction

Validation sur site centralisé

- La technique de prévention de pannes est la journalisation (log)
  - Journal des images avant modification *(Défaire - Undo)*
  - Journal des images après modification *(Refaire - Redo)*
- Technique associée: log write ahead (pré-écriture du fichier journal)
- Structure du fichier journal:
  - identificateur de la transaction
  - identificateur de l'enregistrement
  - le type d'action (insertion, effacement, modification)
  - l'ancienne valeur de l'enregistrement
  - la nouvelle valeur de l'enregistrement
  - pointeur vers l'enregistrement précédent concernant la même transaction
  - les primitives transactionnelles: Début_tr., Valider_tr., Abandonner_tr.
  - Point_de_contrôle contenant les identificateurs des transactions actives
  - ***en réparti*** *- Prépare, Prêt, Abandonner_global, Valider_global, Complet*

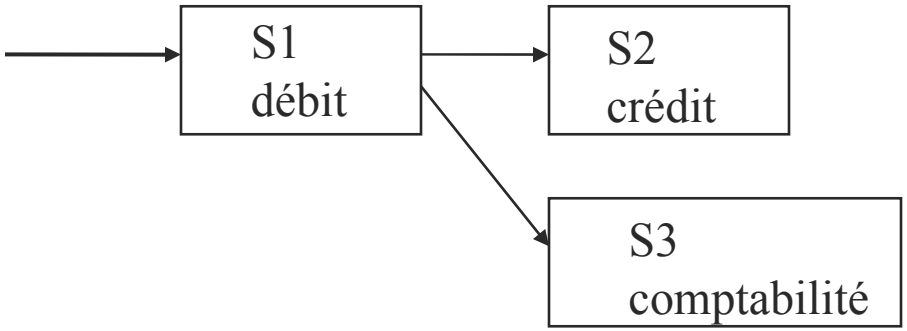
Validation sur site centralisé

- **Procédure de reprise sur site centralisé**
  - ***Pannes sans perte d'information, avec perte de mémoire vive***
    - ① Déterminer toutes les transactions non validées qui doivent être défaites; celles pour qui il y a un Début_tr mais pas de Valider_tr ou Abandonner_tr
    - ② Déterminer toutes les transaction pouvant avoir besoin d'être refaites; celles pour qui il y a un Valider_tr
    - ③ Défaire les transactions identifiées en ① et refaire les transactions identifiées en ②
  - ***Pannes avec perte de mémoire secondaire, avec perte de mémoire stable (fichier journal existe)***
    - Reconstitution de la base en partant de la dernière sauvegarde et en appliquant dessus toutes les images après modification

Transactions réparties

- Une transaction répartie est une transaction ACID dont les parties s'exécutent sur des systèmes différents.

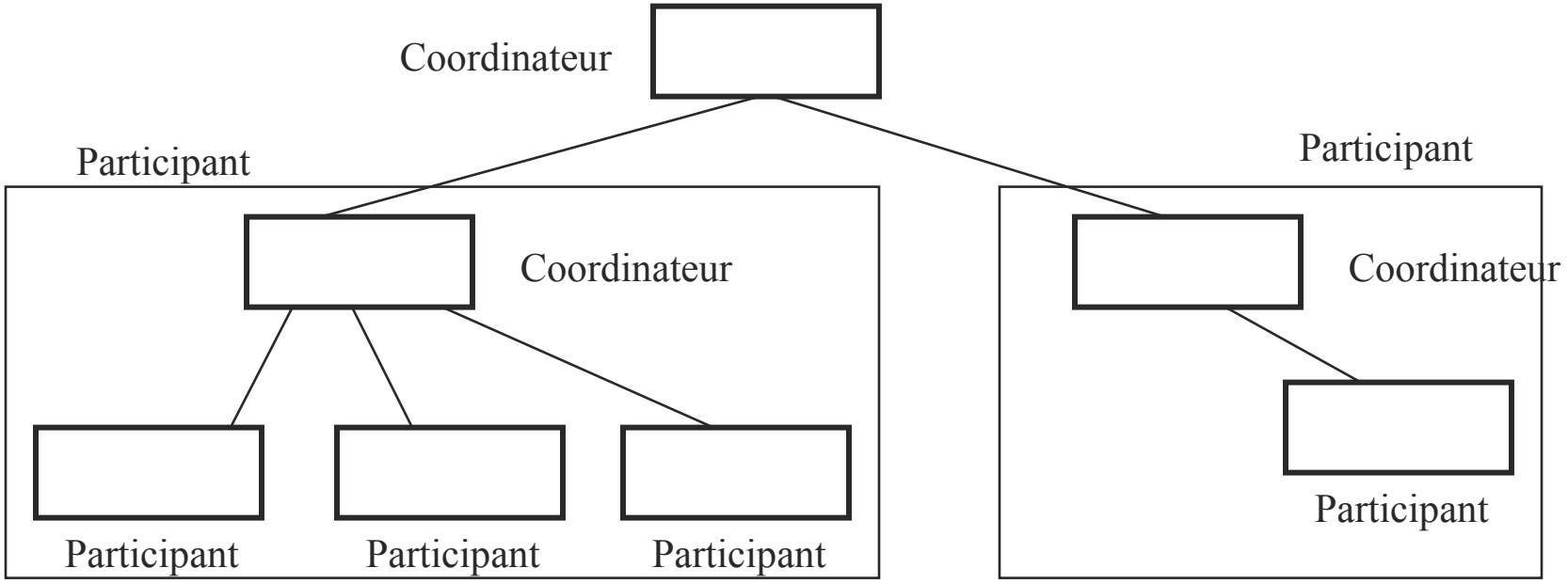
- Exemple: virement bancaire



- Sur chaque site il est possible de mettre en oeuvre la procédure centralisée; le problème se situe au niveau de la décision commune entre les systèmes: *protocole de communication*.

Protocole de validation répartie

- Protocole de validation à deux phases *(Two phase commit)*
- Implémenté dans OSI-TP et SNA-LU6.2
- Structure de communication hiérarchique



Protocole de validation à deux phases

Coordinateur: Ecrire PREPARE dans journal
Envoyer message PREPARE aux participants et activer une temporisation

Participant: Attendre message PREPARE
Si le participant veut valider alors début
Ecrire PRET dans journal
Envoyer PRET au coordinateur
fin
sinon début
Ecrire ABANDONNER dans journal
Envoyer ABANDONNER au coordinateur
fin

Coordinateur: Attendre les réponses (PRET ou ABANDONNER) des participants ou la temporisation
Si chute de temporisation ou au moins une réponse ABANDONNER alors début
Ecrire ABANDONNER_GLOBAL dans journal
Envoyer ABANDONNER à tous les participants
fin

Protocole de validation à deux phases

sinon début
Ecrire VALIDER_GLOBAL dans journal
Envoyer VALIDER aux participants
fin

Participant: Attendre message de commande
Ecrire ABANDONNER ou VALIDER dans journal
Envoyer ACCUSE_DE_RECEPTION au coordinateur
Exécuter la commande

Coordinateur: Attendre ACCUSE_DE_RECEPTION des participant
Ecrire COMPLET dans journal

2PC : Résistance aux pannes

PANNES DE SITE

- Un participant échoue avant d'écrire PRET dans son journal
- Un participant échoue après l'écriture de PRET dans son journal
- Le coordinateur tombe en panne après avoir écrit PREPARE mais avant l'écriture de VALIDER_GLOBAL ou ABANDONNER_GLOBAL
- Le coordinateur tombe en panne après avoir écrit VALIDER_GLOBAL ou ABANDONNER_GLOBAL, mais avant l'écriture de COMPLET
- Le coordinateur tombe en panne après l'écriture de COMPLET

MESSAGES PERDUS

- Un message de réponse (PRET ou ABANDONNER) d'un participant est perdu
- Un message PREPARE est perdu
- Un message de commande (VALIDER ou ABANDONNER) est perdu

RESEAU PARTITIONNE

Que faire ? Timeout…

- Côté coordinateur
  - Timeout en PREPARE
    - Peut choisir d'abandonner
  - Timeout en COMMIT ou ABORT
    - Décision déjà prise > ou renvoie les messages VALIDER_GLOBAL ou ABANDONNER_GLOBAL jusqu'à ACK de tout le monde

Que faire ? Timeout…

- Côté participant
  - Timeout en INITIAL (attente PREPARE)
    - Probablement défaillance du coordinateur > peut choisir d'abandonner.
    - Ensuite, si reçoit PREPARE, soit répond pas, soit vote pour l'abandon
  - Timeout en PRÊT
    - A voté pour commit, ne sait pas ce qu'on fait les autres…
    - Attente (blocage)

Réseau Partitionné ?

- Peut-on décider si la panne provient du réseau ou des participants ?
- Que faire ?
  - En répliqué ?
  - En réparti ?