Les serveurs à architecture parallèle :

Machines Bases de Données (MBdD)

TeraData

Bernard ESPINASSE

Janvier 2018

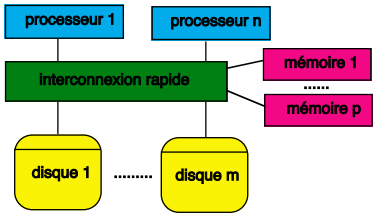
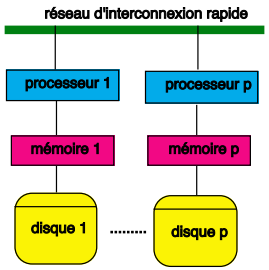
1. Les machines bases de données

- **Typologie d'architectures parallèles**
- **Des SGBDR aux machines base de données**
- **Les avantages attendus des machines bases de données**
- **L'architecture type des MBdD**

Types d'architectures parallèles (1)

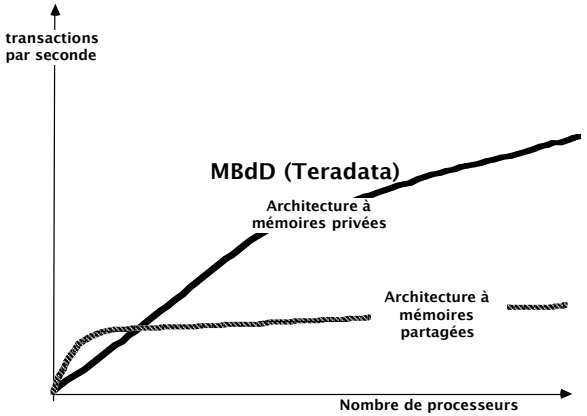
- **Pour augmenter la puissance de calcul d'une machine on peut intervenir sur :**
  - les temps de commutation des circuits de base,
  - la vitesse d'horloge,
  - les transferts d'information (bus d'interconnexion),
  - les accès mémoire,
  - les temps d'exécution des instructions élémentaire,
  - la mise en place de mémoire cache pour alimenter les processeurs en infos,
  - la réalisation parallèle de tâches (OS, SGBD) : du time sharing au parallèlisme de processus (Unix).
- **Processeurs RISC** (à jeu réduit d'instructions) : 35 à 100 MIPS (Millions d'instruction par seconde)
- **Types d'architectures parallèles (depuis 70) :**
  - **Machines parallèles spécialisées** : Machines Base de Données (MBdD) ayant leur propres SGBD,
  - **Machines parallèles génériques** : supportent des adaptations de SGBD du marché parallèlisés (Informix, Ingres, Oracle, Sybase, ...).

Types d'architectures parallèles (2)

Architecture à mémoires partagées *partage tout, couplage fort*	**Architecture à mémoires privées** *partage rien, couplage faible*
	
• composants partagés et banalisés • pb: encombrement du bus de communication → saturation • *machines : Sequent, Sun, DEC, IBM 3090,..*	• ensemble de noeuds coopérant grâce à un réseau d'interconnexion • *machine : MBdD Teradata, ...* *architecture plus **extensible***

Types d'architectures parallèles

- **Performances :**

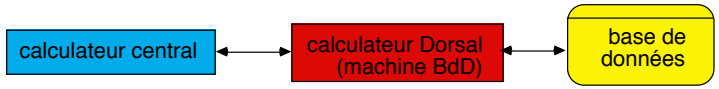


Des SGBD aux Machines Bases de données

- Depuis une dizaine d'années, le volume des BD n'a cessé de croître (plusieurs milliards d'octets)
- Transfert de grands volumes de données en mémoire centrale pour des résultats de faible volume

  -> encombrement de la mémoire et saturation du calculateur par données la plupart inutiles

d'où l'idée de décentraliser les traitements (1979) sur un calculateur spécialisé : DORSAL (machine base de données)



- Dorsal : calculateur classique à logiciel spécifique ou une machine comportant du matériel spécialisé pour la gestion de données
- Algorithmes de recherche et de mise à jour des données exécutés le plus près possible des disques

Avantages des Machines Bases de Données

- **Seules les données utiles sont transférées** vers la mémoire du calculateur central d'où une faible saturation de cette mémoire,
- **Le processeur central ne réalise plus les mises à jour et des recherches → moins sollicité,** il peut s'occuper d'autres tâches pendant que la MBD traite les requêtes envoyées.
- **Exécution des requêtes plus rapide :**
  - la spécialisation du calculateur pour le traitement des bases de données,
  - le développement de technologie de traitement très performante,
  - l'utilisation des machines parallèles pour le traitement des données et aussi pour les accès disques.
- **Grande tolérance aux pannes** car architecture parallèle duplication des processeurs
- **Excellent rapport performance/prix :** baisse des coûts du matériel, algorithmes de parallélisme mieux compris → gains de performance très importants.

L'architecture type des MBD

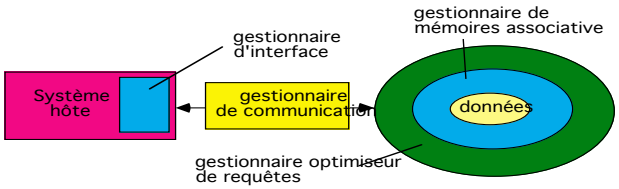
4 couches de fonctions dans une MBD :

1- **gestionnaire d'interface base de données :**
   - prise en compte des requêtes et leur codage
   - localisée sur le système hôte

2- **gestionnaire de communication** : effectuer les échanges entre le calculateur hôte et la machine bases de données (protocole de manipulation de données (TDP))

3- **gestionnaire-optimiseur de requêtes :** analyser et optimiser les requêtes (recherche des chemins d'accès aux données, mise à jour)

4- **gestionnaire de mémoire associative** :
   - effectue la recherche et gère la mémorisation des données
   - est constitué par des processeurs de recherches spécialisés (filtres)



Architecture type des MBD

MBD assurent aussi les fonctions suivantes (leur localisation dépend de la machine considérée) :

- le **contrôle des autorisations d'accès** à la base lors des mises à jour et des consultations
- le **contrôle d'intégrité sémantique** : assurer la cohérence des formats et des types des requêtes envoyées par les utilisateurs
- le **contrôle des accès concurrents** entre les usagers : plusieurs utilisateurs doivent pouvoir accéder à la base sans dommage.
- la **reprise à chaud ou contrôle d'atomicité des transactions** : les requêtes groupées en unités fonctionnelles par l'utilisateur sont soit exécutées totalement soit pas du tout.
- la **restauration à froid de la base** : lors d'une perte d'une partie de la base un mécanisme de restauration permet de récupérer la dernière sauvegarde de cette base.

Plusieurs MBdD sont commercialisées :

- CAFS de ICL,
- DORSAL de Copernique,
- IDM de Britton - Lee
- **DBC/1012 de Teradata.**

2. La machine base de données TeraData

- **Typologie des architectures parallèles**
- **L'architecture**
- **Composants**
- **L'accès aux données, protection, chemin d'accès : index primaires et secondaires**
- **L'évaluation et traitement des requêtes**

Introduction à la MBdD TeraData

- La MBdD Database Computer **DBC/1012** est proposée en 1994 par la société Teradata rattaché à **AT&T** (GIS)
- La société Teradata a été ensuite rachetée, elle était jusqu'en 2007 une division de **NCR Corporation,** la plus grande société de Dayton (Ohio)
- Depuis 2007 la société Teradata Corporation NYSE TDC, est indépendante, c'est un constructeur et éditeur de solutions informatiques spécialisées en matière d'entrepôt de données et d'applications analytiques, dont le siège social de Teradata est maintenant à Miamisburg (Ohio)
- Depuis son origine, la **MBdD Database TeraData** n'a fait qu'évoluer toujours sur les **mêmes principes**,
- Elle repose sur une **architecture parallèle de conception modulaire** permettant l'intégration d'un nombre variable de processeurs (6 à 1024 processeurs)

Utilisation de la TeraData (1)

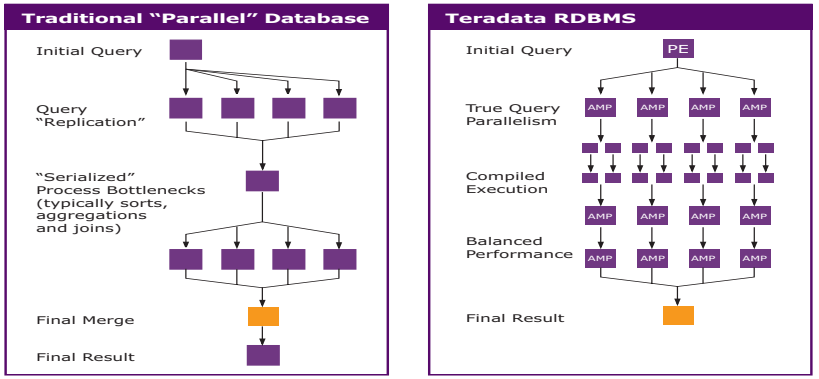
- La MBdD TeraData est utilisée pour gérer de **très groses bases de données relationnelles**, principalement des entrepôts de données (Data Warehouse)
- Teradata a actuellement un peu plus de 1000 clients et plus de 1900 systèmes en production. Parmi les autres clients de Teradata on a :
  - Wal-Mart, AT&T, Bank of America, Barclays, Best Buy, Coca-Cola, Continental Airlines, eBay, FedEx, Verizon, PayPal, T-Mobile et Vodafone, …
- En particulier en **France** :
  - Air France, Armée de terre, Bouygues Telecom, Chronopost, France Télécom, SFR, SNCF, STIF, Crédit Agricole, Banque populaire, BNP Paribas, Caisse d'épargne, Canal+, Leroy Merlin, Groupe Casino, Monoprix (France), ERDF, …

Utilisation de la TeraData (2)

- Le plus gros client est **Wal-Mart**, leader mondial de la grande distribution, qui gère sur Teradata son entrepôt de données d'entreprise : stocks, reporting, applications financières… Ce système est considéré par les experts des Bases de Données comme le **plus grand entrepôt de données au monde** :

Architecture générale de la TeraData (1)

- La **MBD TeraData** repose sur une **architecture parallèle de conception modulaire** permettant l'intégration d'un nombre variable de processeurs (6 à 1024 processeurs), offrant un véritable parallélisme :

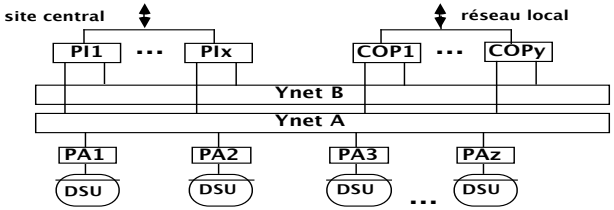


Architecture générale de la TeraData (2)

- Le **système d'exploitation spécifique** des machines Teradata est **NCR UNIX SVR4.2 MP-RAS**, une variante de System V UNIX d'AT&T.
- Cependant, le SGBDR Teradata fonctionne aussi sur des serveurs 64-bit d'Intel sous d'autres systèmes d'exploitation :
  - Microsoft Windows 2000 et Windows Server 2003
  - SUSE Linux Enterprise Server pour les serveurs 64-bit d'Intel.
- Les entrepôts de données Teradata sont alimentés en batch (c'est-à-dire en différé) ou en quasi temps réel grâce à des programmes spécifiques souvent fondés sur des ETL, via ODBC ou JDBC par des applications fonctionnant sous Microsoft Windows ou UNIX.
- La grande majorité des éditeurs du monde décisionnel offre des solutions compatibles avec les systèmes Teradata.

Architecture générale de la TeraData (3)

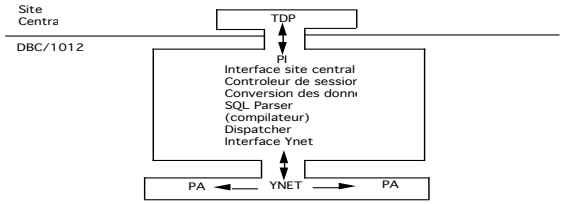
- **la MBD Teradata se compose de :**
  - **Processeurs d'Interface (PI**-InterFace Processor) reliés au site central,
  - **Processeurs d'Accès (PA**-Access Module Processor) sont reliés aux unités de stockage des données, les **DSU** (Disc Storage Unit)
  - d'un système d'interconnexion PI-PA spécifique : **réseau YNET** (bus doublé)
  - **Processeurs de Communication (COP-** Communication Processor) connexion Ynet-Réseau local



- peut être reliée à des ordinateurs centraux : IBM (MVS, VM), Bull, Unisys, NCR,..., via un réseau local à des minis, stations de travail, de PC.
- elle permet l'intégration d'une architecture Client-Serveur

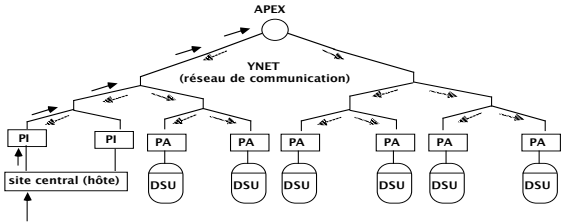
Composants de la TeraData: Processeur d'Interface (PI)

- **Le PI possède 4 composants :**
  - une **interface canal** gére communication avec le calculateur hôte via un canal multiplexeur
  - une **unité centrale** avec un microprocesseur
  - une **mémoire cache associative** de 64 Koct + une **mémoire primaire** de 2 à 8 MOct
  - une **interface YNET** qui gère la transmission des messages sur le YNET.
- **rôle du PI :**
  - vérifier que **requêtes** envoyées par l'ordinateur hôte sont **sémantiquement correctes**
  - les transformer en **séquences exécutables**
  - élaborer un **plan d'exécution**
  - contrôler **l'ordonnancement** et **synchronisation** des différentes **opérations** de ce **plan**

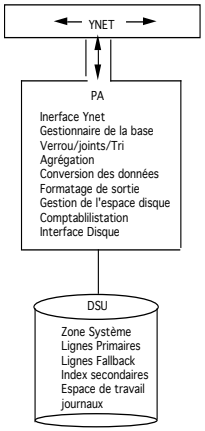


Composants de la TeraData: réseau de communication YNET

- **Ynet** = bus doublé = réseau de communication entre PI et PA permettant le parallélisme
- sécurité et performance : 2 réseaux indépendants **YNET A** et **YNET B**
- **le YNET a une typologie en arbre binaire:**
  - chaque noeud a de la **logique active** pour effectuer des opérations de **tri-fusion**
  - les **feuilles** de ces arbres sont les **PI** et les **PA**
  - les **messages sont fusionnés par le YNET** de sorte qu'un seul message est diffusé à tous les processeurs
  - les 2 réseaux **interconnectent** tous les processeurs et opèrent de façon **concurrente**.



Les composants de la TeraData: Processeurs d'Accès (PA)



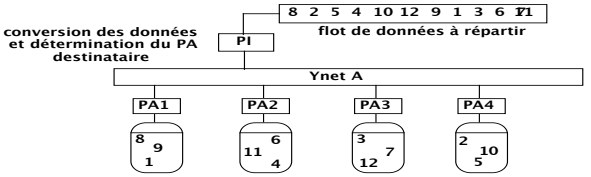
Processeur d'accès = PA

- il exécute les **fonctions de gestionnaire de la base**, d'interface YNET et d'interface disque
- il contient **4 composants** :
  - une **interface YNET** qui gère la communication avec les PI,
  - une **unité centrale** : une mémoire cache de 64 K- octets,
  - une **mémoire primaire de 2 à 8 Mo**,
  - une **interface DSU** qui gère la communication avec les DSU (chaque Processeur d'Accès pouvant être connecté à 2 disques de 500 M- octets à 1,2 G- octets)

- **l'ajout de nouveaux PA** permet un accroissement du nombre de lignes traitées par seconde et les données sont redistribuées vers ces nouveaux PA

L'Unité de Stockage Disque (DSU)

- La répartition des données : **partitionner chaque table** de la BD en **fragments** contenant un nb équivalent de tuples, chaque fragment est **distribué à un PA donné et stocké sur ses DSU associés**
- **tous les PA peuvent exécuter simultanément une même requête**, uniquement sur les données qui leur sont attribuées, le Ynet fera ensuite la fusion des résultats
- **répartition uniforme des données** sur les DSU assurée par **un algo de Hachage :** à partir d'une valeur d'un ou plusieurs attributs (index "primaire") → adresse logique correspondant au lieu de stockage réel sur les DSU = **hash coding sur 32 bits** **(les 4 premiers bits = n° AMP; 8 suivants = emplacement dans l'AMP; bits restants codent l'information du tuple)**



→ du choix de l'index primaire dépend l'uniformité de la distribution :

- plus les valeurs sont uniques, meilleur est le résultat,
- le traitement parallèle est facilité par une bonne distribution.

L'accès aux données (1)

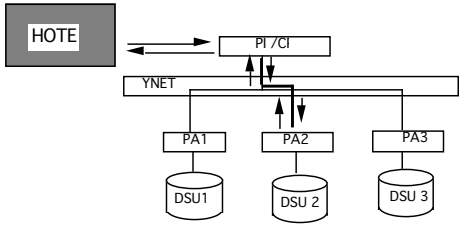
Routage des requêtes sur un seul Processeur d'Accès (PA):

→ **pour les requêtes SQL monotables très restrictives :**

table Employé (Num_Employé , Nom_Employé,....) la clé primaire de cette table est l'attribut Num_Employé

```
SELECT * FROM Employé WHERE Num_Employé = 1008
```

- cette requête SQL précise une valeur unique de l'index primaire (UPI), l'accès se fera suivant le PA qui gère le tuple
- la valeur en hash code de l'index primaire (ici Num_employé) sera utilisée pour accéder aux données.



Accès aux données (2)

Routage des requêtes sur un seul PA :

Les étapes du traitement de la requête sont les suivantes :

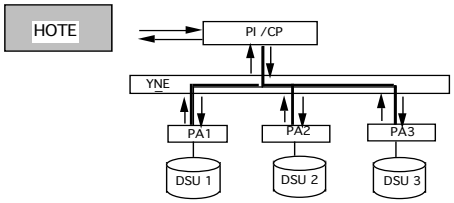
1 - l'utilisateur envoie sa requête exprimée en SQL depuis l'ordinateur hôte,
2 - le Protocole de Transmission de Données (TDP) renvoie ce message vers le PI du DBC,
3- l'analyseur du PI (Processeur d'Interface) vérifie que la requête est sémantiquement correcte et élabore un plan d'exécution,
4- la valeur de l'index primaire est transformée en hash code et est transmise par l'interface YNET du PI au YNET,
5- un message est envoyé par YNET vers tous les processeurs,
6 - l'interface YNET de chaque processeur peut accepter ou refuser le message,
7 - un seul PA va accepter le message,
8 - le message est alors envoyé à ce PA, par le YNET,
9 - le PA va accéder à son DSU et retrouver dans les tables des disques, grâce au hash code le tuple demandé par l'utilisateur,
10 - le PA va construire un message YNET contenant le tuple,
11 - ce message retour envoyé par l'interface YNET sera reçu par tous les processeurs (PI et PA),
12 - mais seul le PI qui avait envoyé la requête accepte le message de retour,
13 - le message lui est envoyé via le YNET/PI,
14 - le PI retourne le message vers l'ordinateur hôte selon le Protocole de Manipulation de Données (TDP),
15 - l'ordinateur hôte est ainsi totalement déchargé du traitement de la requête jusqu'au retour de la réponse.

Accès aux données (3)

Routage des requêtes sur plusieurs PA

Pour requêtes SQL **très restrictives** (porte sur un intervalle de valeurs de l'Index Primaire (Employe_number)) :

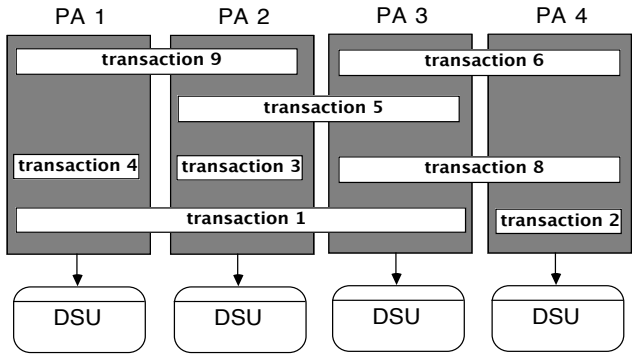
```
SELECT Nom_Employé, Num_Employé  FROM Employé
     WHERE Num_Employé BETWEEN 1001 AND 1010  ORDER BY Nom_Employé
```



1 - le compilateur SQL du PI génère un traitement sur plusieurs PA,
2 - le YNET transmet le message à tous les processeurs,
3 - l'interface YNET de chaque PA accepte le message,
4 - chaque PA va accéder à la table sur ses DSU et copier les tuples qui correspondent à la réponse dans son espace de travail (DataSpool). Un message ensuite émis sur le réseau YNET,
5 - l'interface YNET du PI origine collecte les réponses fusionnées et déjà triées dans un buffer,
6 - le PI envoie ensuite le contenu de son buffer au TDP de l'ordinateur hôte,

Parallélisme dans une DBC/1012

- **Plusieurs processeurs travaillent en // sur une même requête**
- **Plusieurs processeurs travaillent en // sur plusieurs requêtes**

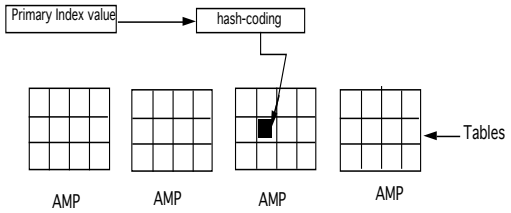


Protection des données

- **Le fallback**
  - copie des tuples de la base dans 2 DSU différents → redondance des tables
  - permet la protection des données pendant la panne d'un PA
- **Regroupement de fallback (Fallback clustering)**
  - un groupement (cluster) peut contenir de 2 à 16 PA
  - la panne d'un PA dans un cluster n'a pas d'effet sur la disponibilité des données, car les données sont dupliquées dans un autre PA du regroupement
  - mais la panne de 2 PA dans un même regroupement entraîne automatiquement un arrêt du système
  - le clustering permet de minimiser les risques d'arrêt de fonctionnement du système.

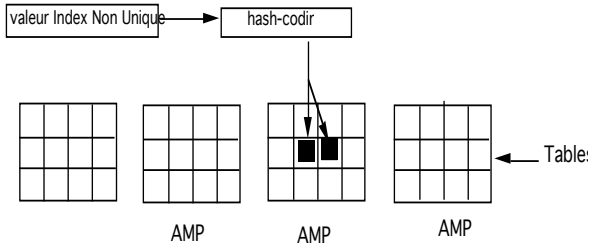
Les index primaires PI : UPI & NUPI

- Définition **obligatoire** pour chaque table → définit sa structure de stockage (par défaut 1° attribut)
- **choisi en fonction des utilisations de la base** (mode décisionnel - peu de mises à jour- ou transactionnel - mises à jour fréquentes) et de la fréquence des requêtes
- peuvent être : **uniques** (pas de duplication de tuples autorisé) : **UPI** ,
  ou **non uniques** : **NUPI**.
- **requête portant sur UPI** : utilise **un seul AMP** et le résultat retourné = **un seul tuple** (les accès par index primaire ne nécessite aucun fichier intermédiaire) :



Les index primaires PI : NUPI

Requête portant sur un **NUPI** : certaines **valeurs de l'index primaire sont dupliquées** et les hash-codes de plusieurs tuples sont les mêmes, les tuples dupliqués sont stockés dans le même AMP :



- une requête sur NUPI mettra en oeuvre **un seul AMP**
- mais le résultat de la requête peut être constitué de **un ou plusieurs tuples**
- dans ce cas il peut exister un **fichier spool** dans lequel les résultats intermédiaires sont stockés.

Les index secondaires SI : USI & NUSI

- Autre façon de sélectionner les tuples dans les tables, chemins d'accès souvent utilisés
- la RDBC construit pour chaque index **secondaire unique** (**USI**) ou **secondaire non unique** (**NUSI**) une **sous-table** :
  - elle contient l'ensemble des index secondaires codés suivant un algorithme de hachage
  - à chaque index de cette sous-table correspond un identificateur = valeur hash-code de l'index primaire, permet de retrouver le tuple à sélectionner dans les DSU
  - chaque ligne d'une sous-table pointe sur une ligne de la table de la base

taille tuple	identificateur de l'index secondaire	Valeur de l'index secondaire	identificateur de l'index primaire	taille tuple

Structure de hachage des index secondaires

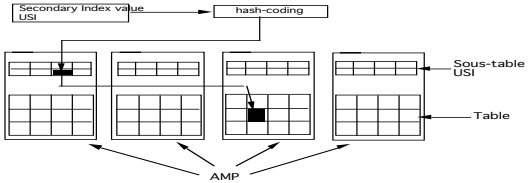
Les index secondaires SI : USI & NUSI

Usage des index secondaires :

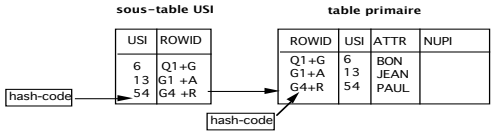
- peut avoir des **coûts non négligeables pour les opérations d'INSERT, UPDATE et DELETE**
- **occupent un espace mémoire** qui peut être très important, en effet ils nécessitent un stockage particulier dans tous les AMP
- **leur modification entraînent non seulement des modifications** sur les **tables** mais aussi des modifications sur les **sous-tables** de la base
- en mode transactionnel avec nécessité de faire des mises à jour, les index secondaires peuvent être **très pénalisants** pour les temps de réponse car les volumes à réorganiser deviennent alors beaucoup plus importants.

Les index secondaires uniques (USI)

- La valeur de l'index secondaire est transformée par hachage et dans chaque AMP le système crée des sous-tables dans lesquelles sont stockées ces index secondaires :



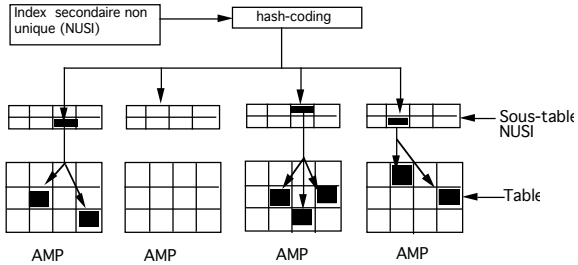
- Ces sous-tables possèdent un colonne d'identificateur de lignes primaires. A chaque valeur de cette sous-table correspond un et un seul tuple :



- Permet un accès rapide aux données et 2 AMP sont au maximum sollicités pour répondre à la requête (tableau), un seul tuple est retourné. le bon choix des indices secondaires uniques est un excellent moyen pour optimiser une table.

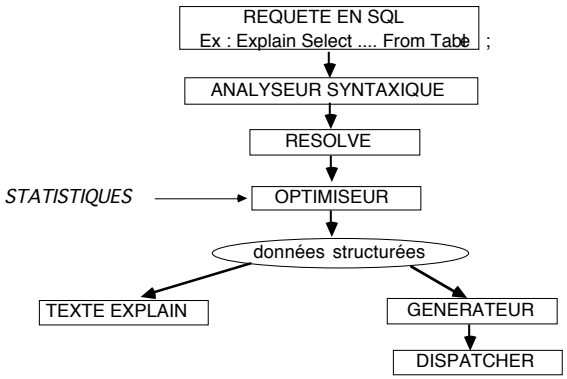
Les index secondaires non uniques (NUSI)

- **Idem USI** , mais certains tuples indexés pouvant être **dupliqués**
- sont placés dans des **sous-tables d'index secondaires**, dans ces sous-tables des identificateurs de tuples (rowid) renvoient à des valeurs dans les tables primaires de la base
- lorsqu'on envoie une requête sur un NUSI **toutes** les sous-tables index secondaires de tous les AMP sont accédées :



Structure de stockage des NUSI

Evaluation et optimisation des requêtes



- L'évaluation des requêtes est réalisée par un optimiseur (Parser des IFP)
- 2 commandes utilisées : **COLLECT STATISTIC** et **EXPLAIN**

La commande COLLECT STATISTICS

- L'optimiseur utilise les **données statistiques** recueillies sur les tables pour rechercher les meilleurs chemins d'accès :
  - le **nombre de tuples de la table**,
  - les **index uniques et non uniques, primaires ou secondaires**,
  - le **nombre de valeurs par attributs**,
  - les valeurs **maximum** et **minimum** dans chaque attribut de la table,
  - le **nombre d'occurrences non dupliquées** dans chaque table
- Le recueil de statistiques ne s'effectue que sur demande de l'utilisateur. Si elle n'est pas faite, le système fera un échantillonnage dynamique et choisira un AMP au hasard pour faire ses statistiques.

  syntaxe : {COLLECT} STATISTICS [ON] table X [COLUMN x] [INDEX x [..., x]] .

  Exemple : COLLECT STATISTICS ON CLIENT Column clsurfin;
- Elle doit être faite chaque fois que le nb tuples affectés par INSERT, UPDATE, DELETE dans une table est > à 10% des tuples de la table, ou à la création des tables de la base
- DROP STATISTICS ON table X [COLUMN x] [INDEX x [..., x]] doit précéder toute nouvelle instruction de collecte de statistiques.
- En transactionnel les coûts d'un rafraîchissement de statistiques peuvent être importants, mais sont indispensables pour un bon choix des chemins d'accès.

La commande EXPLAIN

- Placée au début d'une requête permet d'obtenir un **compte-rendu d'une simulation d'exécution de la requête** construite par l'optimiseur grâce à ses statistiques.
- La commande **Explain** renseigne sur :
  - la nature des tables sollicitées pour traiter la requête,
  - les index utilisés (secondaires, primaires),
  - les différentes étapes du traitement de la requête
  - le contenu des fichiers créés pour stocker les résultats intermédiaires (Spool)
  - le nombre de tuples retournés à chaque étape du traitement,
  - les temps d'exécution des différentes étapes de la requête,
  - les types de jointures
  - le temps total mis par la RDBC pour exécuter la requête.

Attention ! les résultats fournis par l'Explain concernant une requête :

- ne sont que des **estimations**
- ne **correspondent pas toujours aux valeurs exactes** que donnerait une exécution de la requête.

Exemple de compte rendu d'Explain

On veut compter le nombre d'adresses distincts de personnes ayant une surface financière déterminée et des types de compte bien précis dans la base de données "Nouvelle".

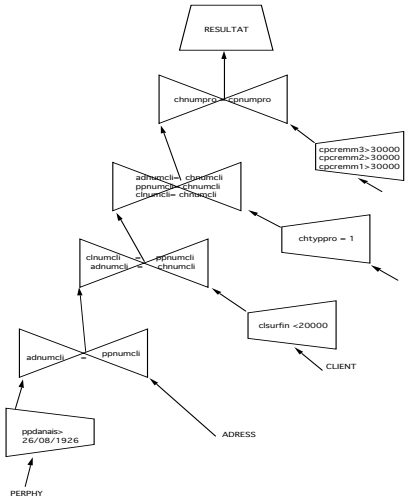
Expression de la requête :

```
EXPLAIN COUNT (DISTINCT ADLIADR1)
FROM ADRESS, CLIENT, PERPHY, CHOISI, CCPCOM
WHERE ADNUMCLI=CLNUMCLI
AND CLNUMCLI= PPNUMCLI
AND PPNUMCLI=CHNUMCLI
AND CHNUMPRO=CPNUMPRO AND CHTYPPRO=1
AND CPCREMM1>3000 AND CPCREMM2>3000 AND CPCREMM3>3000
AND CLSURFIN >20 000
AND PPDANAIS BETWEEN '26/08/1926' AND '26/08/1931';
```

Le compte rendu Explain nous renseigne sur :

- le temps met le système pour balayer la table PERPHY,
- les jointures et les restrictions nécessaire à l'exécution de la requête,
- le temps que met le système pour faire ses jointures,
- le nombre de tuples mis en jeu,
- le nombre de tuples retournés.

Arbre algébrique associé à la requête



Traitements des requêtes sur la RDBC : Requêtes mono-table

Plusieurs types de traitement des requêtes suivant la nature des index utilisés :

- **les requêtes sur UPI** : → 1 seul AMP est accédé et va chercher dans son DSU le tuple qui répond à la requête.
- **les requêtes sur NUPI** : → 1 seul AMP est accédé les tuples indexés dupliqués sont placés dans le même AMP. L'AMP va chercher dans ses DSU le tuple ou les tuples répondant à la requête.
- **les requêtes sur USI :** → 2 AMP sont accédés au maximum, dans le premier le système va chercher les tuples de la sous table USI , et dans le deuxième AMP il va trouver le tuple de la table répondant à la demande et ceci grâce au identificateur de tuples de la sous-table.
- **les requêtes sur NUSI** : → **tous** les AMP sont sollicités, le système va chercher dans la sous-table NUSI de chaque AMP les identificateurs de tuples qui lui permettront de retrouver dans les tables les tuples.
- **les requêtes portant sur des attributs non indexés :** → **tous** les AMP sont sollicités et le système ne passe plus par les sous-tables. Le système fait un balayage séquentiel complet de la table (Full Table Scan ; FTS).

Traitements des requêtes sur la RDBC : Requêtes multi-tables

- Les requêtes faisant appel à des données provenant de plusieurs tables sont traitées par des **jointures**
- Pour qu'il puisse y avoir une jointure entre deux ou plusieurs tables il est nécessaire que les tables aient au moins un attribut en commun
- La RDBC utilise **5 types de jointures** pour traiter les requêtes multitables :
  - **NESTED JOIN**
  - **MERGE JOIN**
  - **PRODUCT JOIN**
  - **EXCLUSION JOIN**
  - **ROWID JOIN**
- Le type de jointure utilisé dépend de la **nature des index et des statistiques** collectées sur les tables de la base
- Certaines jointures peuvent être très pénalisantes :
  - si **mauvais choix** des index primaires et secondaires
  - si **pas de mise à jour récente des statistiques**
- Dans la RDBC de 2 à 16 tables peuvent participer à des jointures dans une requête.

Les NESTED JOIN

- **Jointures les plus efficaces dans la RDBC** : le nombre d'AMP impliqué dans une Nested join dépend de la nature des attributs participant à la jointure :
- Possibles que sous certaines conditions :
  1- la condition WHERE spécifie une valeur **constante** pour un **index unique (primaire ou secondaire)**
  2- un attribut du tuple déterminé par la condition précédente soit **égalé** avec un attribut index primaire ou secondaire (unique ou non unique) de la deuxième table.

Exemple : soient les tables TITULA et CLIENT suivantes :

TITULA

Tinumcli	Titypdet	Tinumpro
PK		FK
UPI		

CLIENT

Clnumcli	Clposccp	clsurfin
PK		FK
UPI		

avec Clnumcli : UPI ou USI,Tinumpro : index primaire ou secondaire

soit une requête avec une jointure de type nested join :

```
Select Clposccp, Titypdet FROM Titula , Client
    WHERE Clnumcli = Tinumpro  And Tinumcli = '120154';
```

- *Une Nested Join sur uniquement des index secondaires ne fait appel qu'à 2 AMP.*

Les MERGE JOINT

Types de jointure assez courantes, en réalité il existe 3 types de Merge Join :

1 - les MERGE JOIN sans redistribution

Requêtes de la forme suivante :

```
Select... FROM Service, Employé WHERE Service N° Serv = Employé N°_Empl;
```

Service

N° Serv	Nom_Serv	Ville	N°_Dir
PK			FK
UPI			

Employe

N°_Empl	Nom	N°_Serv
PK		FK
UPI		

- La jointure est faite entre **2 UPI**
- Type de **jointure très efficace** car ne nécessite pas de redistribution les valeurs hache code des attributs sont comparés avec les valeurs hache code des tuples placés dans le fichier Spool
- Les **tuples sélectionnés sont places dans un autre fichier Spool** et renvoyés à travers le Ynet à l'utilisateur.

Les MERGE JOINT

2 - les MERGE JOIN avec redistribution d'une des tables

- Les tuples d'une des tables (accédée par un balayage complet de la table) sont hachés et redistribués sur les AMP
- Il y a ***redistribution complète de l'une des deux tables participant à la jointure*** (cf figure suivante).

Les requêtes avec redistribution d'une des tables sont de la forme :

```
SELECT... FROM table T1, table T2   WHERE table T1.col_A = table T2.col_B;
```

avec col_A = index primaire et col_B = non index primaire.

- Les MERGE JOIN de ce type nécessitent que la **jointure se fasse entre un attribut INDEX PRIMAIRE** et un **attribut non index primaire d'une autre table**. Il y a création d'un **fichier spool** pour la redistribution des tuples.

Les MERGE JOINT

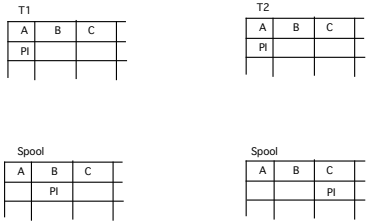
3 - les MERGE JOIN avec redistribution complète des tables

Si la jointure est faite sur des **attributs non index primaires** on a une redistribution complète des deux tables sur les colonnes de jointures

Requêtes de la forme :

```
SELECT... FROM table T1, table T2  WHERE table T1.col_B= table T2.col_ C;
```

avec col_C = non index primaire et col_B= non index primaire.



c'est la forme des MERGE JOIN **la plus coûteuse**, donc on cherchera à l'éviter en exploitant les compte-rendus de la fonction **explain** pour **réécrire la requête**.

Les PRODUCT JOINT

- Jointures les plus **courantes**
- *Utilisés chaque fois que la **restriction n'est pas basée sur une égalité** ou que l'optimiseur **ne peut pas utiliser** des **Merge Join** ou des **Nested Join***
- Chaque tuple d'une table est comparé à tous les tuples de l'autre table, le système ne fait pas de hachage sur les attributs des tables

Requêtes de la forme :

```
SELECT... FROM table X, table Y
     WHERE table X.PI = 100 AND table Y.col_1 < > table Y.col_1;
```

avec col_1 = non index primaire

- le nombre de comparaisons à effectuer dans une jointure PRODUCT JOIN est très important il est égal au :

nbre tuples de la plus petite table X nbre tuples de la plus grosse table.

- seuls les tuples retenus à l'issu de cette comparaison sont joints et retournés à l'utilisateur. Cette comparaison est très couteuse en terme d'espaces disque à cause du nombre très élevé de comparaisons qu'il faut réaliser

Les CARTESIAN PRODUCT

- Le produit cartésien = forme de **PRODUCT JOIN** mais **sans aucune restriction**.
- La plus petite des 2 tables participant à une jointure est intégralement dupliquée sur chaque PA
- Les **résultats intermédiaires** sont placés dans des **fichiers Spool**
- Les jointures PRODUCT JOIN soient très coûteuses en terme d'espaces disques et on doit les éviter

Requêtes de la forme :

```
SELECT Clnumcli, Chtyppro  FROM Client, Choisi;
```

- Aucune restriction dans une CARTESIAN JOIN, le nombre de tuples retournés est égal au :

nbre tuples de la plus petite table X nbre tuples de la plus grosse table.

- La jointure sur le produit cartésien est **peu utilisée** car elle **consomme énormément de ressources** et les requêtes ce type **avortent souvent** car le fichier spool est souvent débordé

Les EXCLUSION JOINT

- Jointures des requêtes avec l'instruction **NOT IN** : il s'agit en fait ***d'éliminer les tuples qui ne répondent pas à la restriction***
- **Nécessitent un balayage complet** des tables c'est une **jointure très coûteuse**

Requêtes de la forme :

```
SELECT Ppdanais, Clsurfin  FROM PERPHY, CLIENT
 WHERE Clnumcli = 1225632  And Cl NOT IN (Select sale_num From table Y);
```


Les ROWID JOINT

Ce sont des jointures particulières, elles requièrent 2 conditions :

- une valeur constante pour l'index NUPI
- une sélection peu restrictive pour le NUSI

Requêtes de la forme :

```
SELECT * From tableX, table Y Where tableX.col_1=10 And table X.col_2=tableY.col_
```

avec col_1 = Index Primaire Non Unique et col_2 et col_3 Index Secondaire Non Unique

- les **tuples de la table X répondant à la requête** sont **dupliqués dans tous les PA**.
- les valeurs de l'attribut de jointure de la table X sont **hachées selon les NUSI de la table Y**
- les **tuples sélectionnés dans les sous-tables** sont placés dans un **fichier Spool** pour **chaque PA**
- les **fichiers spool** sont ensuite **triés**
- ces **fichiers sont ensuite joints avec** les **tuples sélectionnés** dans la **table X.**