Module Bases de Données, ESSI3
Université de Nice et Sophia-Antipolis

SGBDs parallèles et distribués : Architecture et Algorithmique

Support de cours
Décembre 1994

N. Le Thanh

Table des matières

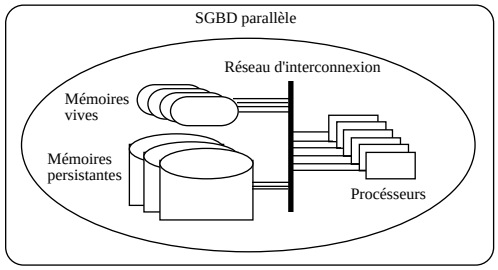
I- INTRODUCTION

- Notion de SGBDs parallèles
- Problèmes à étudier dans les SGBDs parallèles
- Notion de SGBDs distribués
- Problèmes à étudier dans les SGBDs distribués

I- INTRODUCTION

1- NOTION DE SGBDs parallèles

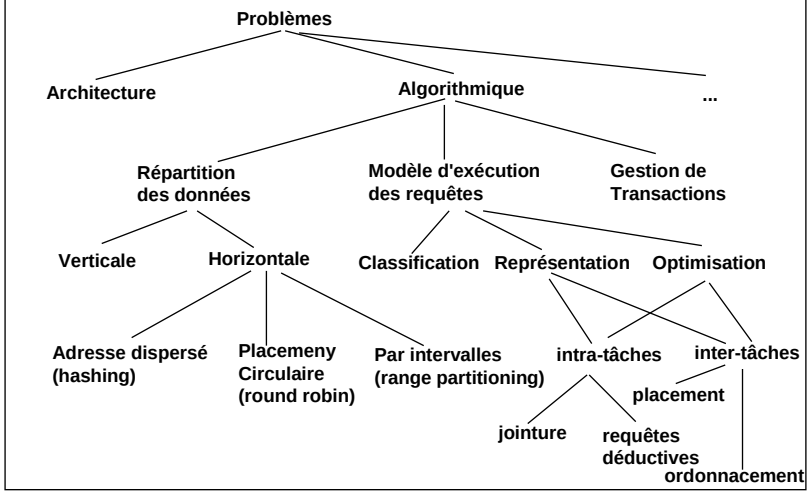
• **Machnine BD : Un SGBD implanté sur une machine à architecture parallèle**



• **Objectifs : Performance - Sécurité - Extensivité - Disponibilité des données - Prix**

I- INTRODUCTION

2- Problèmes à étudier dans les SGBDs parallèles

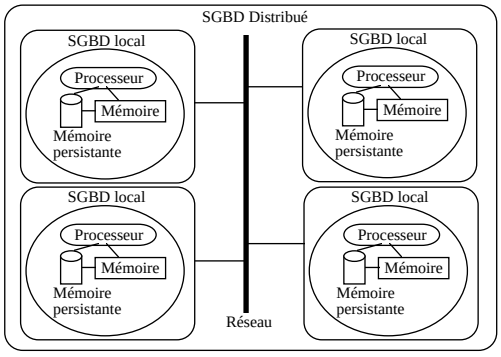


- 5 -

I- INTRODUCTION

3- Notion de SGBDs distribués

SGBD distribué : Interconnexion de plusieurs SGBD locaux

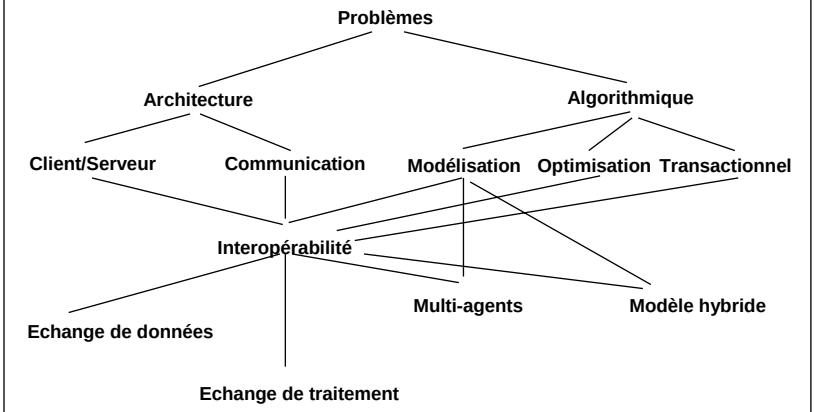


Objectifs : Echange de données et de traitements à distance - Applications réparties

- 6 -

I- INTRODUCTION

4- Problèmes à étudier dans les SGBDs distribués

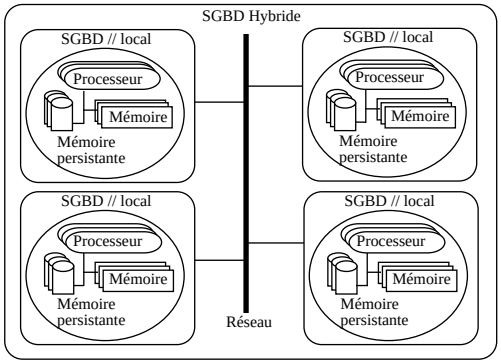


- 7 -

I- INTRODUCTION

5- Notion de SGBDs Hybrides

SGBD hybride : Interconexion des SGBDs parallèles



Objectifs : Avantages des deux types de systèmes

- 8 -

II- SGBDs PARALLELES

- Classification des machines BD
- Modélisation de requêtes parallèle
- Algorithmes intra-opération
- Algorithmes iner-opération

II- SGBDs PARALLELES

1- CLASSIFICATION DES MACHINES PARALLÈLES

1.1- Classement "parallèlisme"

- S.I.M.S. (Single Instruction Multiple Data)
  - grain fin
  - parallèlisme massif
  - exécution synchrone sur un gran nombre de processeurs
  - Ex. : Connexion Machine - Réseau de neurones

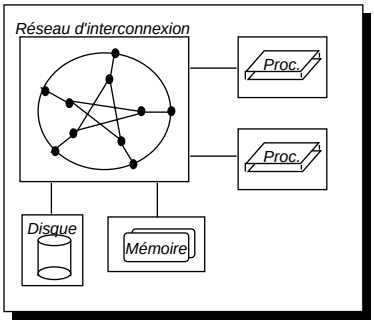
- S.P.M.D. (Single Programme Multiple Data) et M.I.M.D. (Multiple Instructions Multiple Data
  - grain moyen
  - nombre limité de processeurs (10 à 1000)
  - exécution et communication asynchronne possible
  - Ex. : Computing Surface de Meiko - Réseau de transputeurs

- M.P.M.D. (Multiple Programmes Multiple Data)
  - gros grain
  - peu de processeurs
  - parallismre au niveau des applications
  - pas de l'horloge globale - mécanismes de synchrone classiques
  - Ex. : Windows-NT à processeurs symétriques

II- SGBDs PARALLELES

1- CLASSIFICATION DES MACHINES PARALLÈLES

1.2- Classement "Base de données"



Système à mémoire partagée

*** Avantages :**
- mise en oeuvre simple du parallélisme inter-requêtes
- bonne utilisation des ressources
- équlibrage automatique des charges
- communication efficace

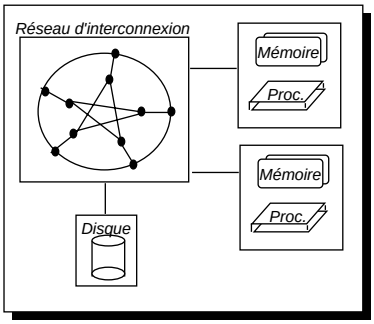
*** Inconvenients**
- taille limitée (10 à 20 processeurs)
- accéssibilité aux données limitée
- prix est un peu élévé

*** Exemples des systèmes :**
- DB2, ORACLE sur IBM 3090, Bull DPS8
- Oracle, Ingres, Sybase sous Unix sur Sequent, Encore, ...
- XPRS de Berkeley, DBS3 de Bull-Inria

II- SGBDs PARALLELES

1- CLASSIFICATION DES MACHINES PARALLÈLES

1.2- Classement "Base de données"



Système à disques partagés

*** Avantages :**
- Taille moyenne (100 processeurs)
- prix faible
- bon équlibrage des charges
- dublication automatique des données fréquemment utilisées
- bonne accéssibilité aux données
- résistance aux pannes

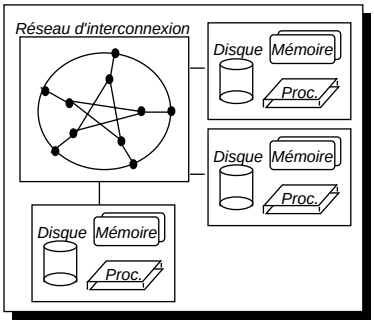
*** Inconvenients**
- cohérence des copies multiples
- saturation du réseau à cause de transfert de données
- bande passante de données limitée

*** Exemples des systèmes :**
- IBM : IMS/VS, Data Sharing Product
- DEC : Vax DBMS, Vax RDB/VMS, Vax Cluster, Oracle sur NCUBE

II- SGBDs PARALLELES

1- CLASSIFICATION DES MACHINES PARALLÈLES

1.2- Classement "Base de données"



Système à mémoire distribuée

*** Avantages :**
- **Taille extensible (millier processeus)**
- **prix moins cher**
- **bonne accéssibilité aux données**
- **bonne résistance aux pannes**

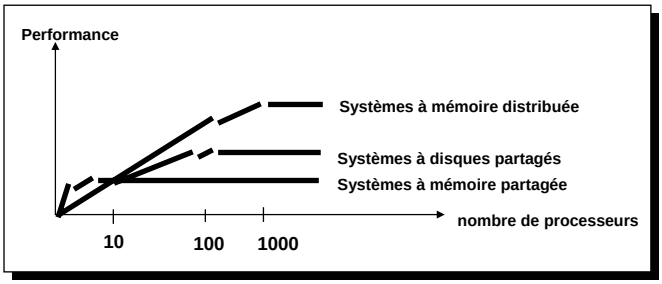
*** Inconvenients**
- **difficile à administrer**
- **répartition de données**
- **équilibrage de charge difficile**
- **portage difficile**

*** Exemples des systèmes :**
- **TANDEM : NonStop SQL,**
- **Téradada DBC 1012**
- **Univ. du Wisconsin : GAMM, MCC: Bubba, Esprit EDS**

II- SGBDs PARALLELES

1- CLASSIFICATION DES MACHINES PARALLÈLES

1.2- Classement "Base de données"



II- SGBDs PARALLELES

2- MODÈLISATION DES REQUÊTES PARALLÈLES

2.1- Classification des requêtes

- Requêtes décisionnelles (desision support)
- **gros grain (nécessite une puissance de calcul - coarse grain parallelism)**
- **en lecture pour but une expertise des données**
- **masse de données importantes et imprévisible**
- **banc d'essai Wincosin Benchmark**

- Requêtes OLTP (On Line Transaction Processing)
- **grain moyen**
- **consultation et mise à jour**
- **bancs d'essai TCP-A et TCP-B**

- Requêtes déductives
- **grain moyen**
- **déduction ou contraintes d'intégrité**
- **banc d'essai TCP-C**

- Requêtes navigationnelles
- **grain fin**
- **recherche et mise à jour courte 5CAO / FAO)**
- **banc d'essai 001 et 007**

II- SGBDs PARALLELES

2- MODÈLISATION DES REQUÊTES PARALLÈLES

2.2- Principe d'exécution parallèle

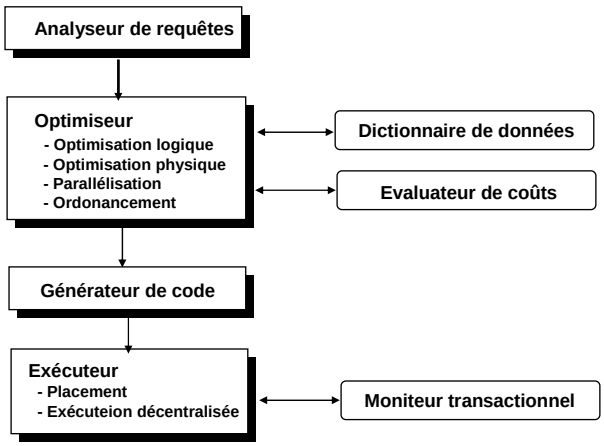
- Transaction :
Une transaction est un ensemble de tâches à exécuter qui respecte les propriété suivantes :
- **Atomicité : soit toites les tâche sont bien exécutées soit rien n'est exécutée**
- **Cohérence : l'exécution d'une transaction respecte toutes les contraintes d'intégrité**
- **Isolation : les résultats intermédiaires d'un transaction ne sont pas visibles pour les autres transactions**
- **Durabilité : Les résultats d'une transaction validée est persistant**

- Transaction parallèle :
Une transaction parallèle est une transaction dont les tâches sont exécutées en parallèle

- Une requête sur un SGBD parallèle se traduit en général en une transaction parallèle

II- SGBDs PARALLELES

2- MODÈLISATION DES REQUÊTES PARALLÈLES
2.2- Principe d'exécution parallèle



- 17 -

II- SGBDs PARALLELES

2- MODÈLISATION DES REQUÊTES PARALLÈLES
2.3- Représentation des requêtes parallèles

a) Eléments de base d'une transaction (requête) à représenter :

- **les tâches et leur durée d'exécution**
- **les dépendances temporelles entre ces tâches : les synchronisations, les relations de précédence, ...**
- **les volumes de données traitées par chaque tâche et la taille de leur résultat**
- **les techniques de communication des résultats entre les tâches**
- **les contraintes de placement si elles existent**

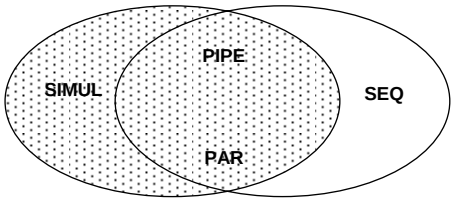
- 18 -

II- SGBDs PARALLELES

2- MODÈLISATION DES REQUÊTES PARALLÈLES

2.3- Représentation des requêtes parallèles

b) Les synchronisations :



Parallélisable **Séquentielisable**

SIMUL : les tâches coopératives

PIPE : les tâches en mode comsomateur / producteur

PAR : les tâches indépendantes

SEQ : les tâches en séquence

START, END : Début et fin de la transaction

II- SGBDs PARALLELES

2- MODÈLISATION DES REQUÊTES PARALLÈLES

2.3- Représentation des requêtes parallèles

c) Graphe de tâches :

- Graphe connexe, acyclique dont les noeuds représents des ttâches et les arcs sont marqués par des synchrolisations

*** Graphe d'exécution : graphe de tâches qui détaille le parallélisme intra-tâche**

*** Graphe de contrôle : graphe des marques de synchrolisation détaillée**

- as (attentr simple) : attente de la réception du message de fin de la tâche précédente

- am (attente multiple) : attente de la réception du message de fin de toutes les tâches précédentes

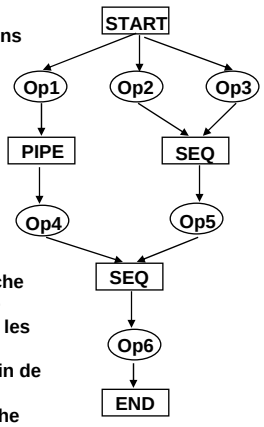
- al (attente locale) : indique l'attente locale de la fin d'une tâche

- sd (déclenchement simple) : ordone l'exécution d'une tâche

- md (déclenchement multiple) : ordone l'exécution de toutes les sous-tâches d'une tâche

- sp (propagation simple) : envoi d'un message indiquant la fin de la tâche vers la tâche suivante

- mp (propagation multiple) : message indique la fin de la tâche vers toutes les tâches suivantes



II- SGBDs PARALLELES

2- MODÈLISATION DES REQUÊTES PARALLÈLES
2.3- Représentation des requêtes parallèles

d) Attributs sémantiques des graphes :

- Volumes de données en entrée et en sortie des tâches
- modèles analytiques

- Temps d'exécution des opérateurs
- calculé par les modèles analytiques avec les volumes d'entrée comme paramètres

- Méthodes de communication des résultats ---> Coût de communication
- DISTR : distribution (fonction de distribution de données)
- DIFFU : diffusion
- PROPA: Propagation (point-à-point)

- Contraintes de placement physique des tâches
- Contraintes fixes (localité de stockage statique de données)
- Contraintes variables (localité dynamique de données)

- 21 -

II- SGBDs PARALLELES

2- MODÈLISATION DES REQUÊTES PARALLÈLES
2.3- Représentation des requêtes parallèles

e) Temps de réponse d'une requête :

$fOp_i(j,N)$: fonction de coût O sur j processeurs et N données

Temps de réponse globale — **Temps de réponse locale**

Temps de réponse économique

$t_{éco}(Op_i, N, ²) = \min fOp_i(p_{éco}(Op_i, N, ²), N)$

Temps de réponse optimal

$t_{opt}(Op_i, N) = \min fOp_j(j,N)$

Nombre de processeurs économique

$p_{éco}(Op_i,N,²) = \min\{p/\ \delta(fOp_j(p,N), t_{opt}(Op_i, N), ²)\}$

Nombre de processeurs optimal

$p_{opt}(Op_i,N) = \min\{p/\ fOp_j(p,N) = t_{opt}(Op_i, N)\}$

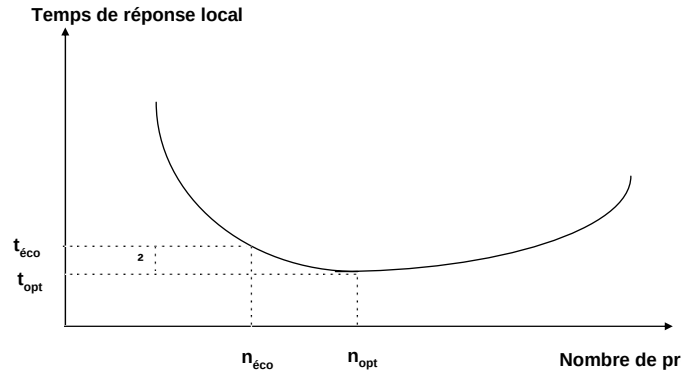
$$\delta(a,b,²) = \begin{cases} a\text{-}b\ Š\ ² \\ (a\text{-}b)/b\ Š\ ²/100 \end{cases}$$

- 22 -

II- SGBDs PARALLELES

2- MODÈLISATION DES REQUÊTES PARALLÈLES
2.3- Représentation des requêtes parallèles

e) Temps de réponse d'une requête :

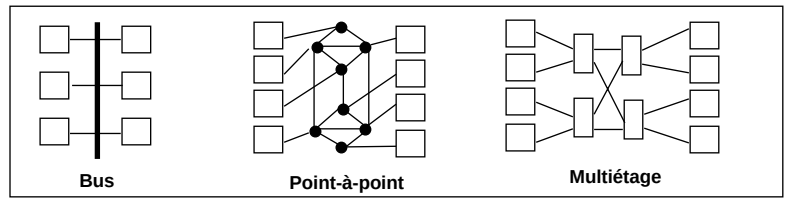


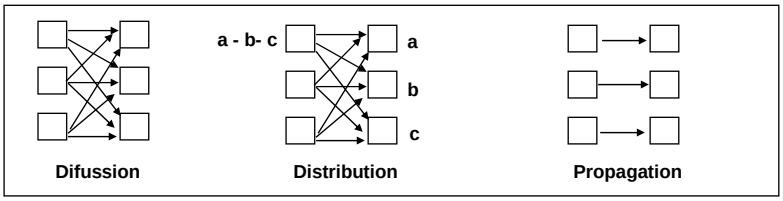
- 23 -

II- SGBDs PARALLELES

2- MODÈLISATION DES REQUÊTES PARALLÈLES
2.3- Représentation des requêtes parallèles

e) Types de communication ::



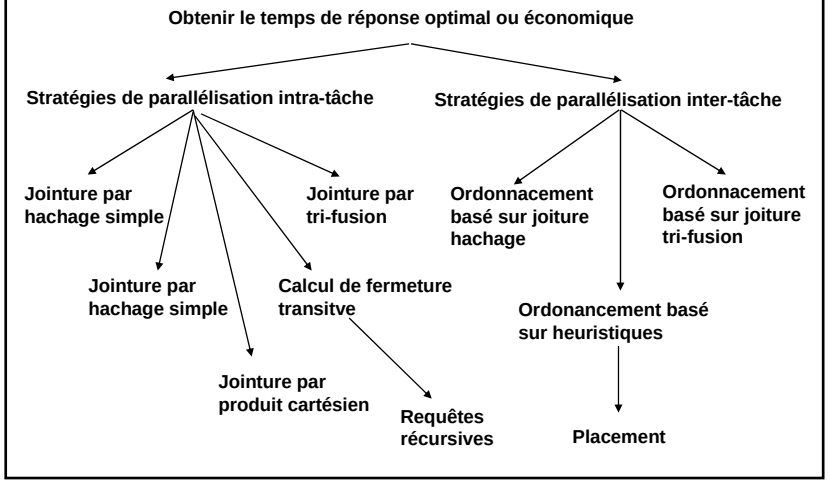


- 24 -

II- SGBDs PARALLELES

2- MODÈLISATION DES REQUÊTES PARALLÈLES

2.4- Objectif et stratégies

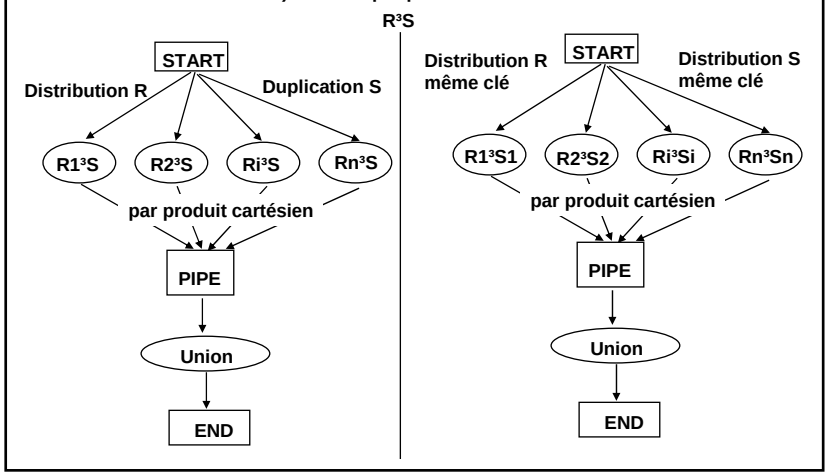


II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.1- Algorithmes de jointure

a) Jointure par produit cartésien



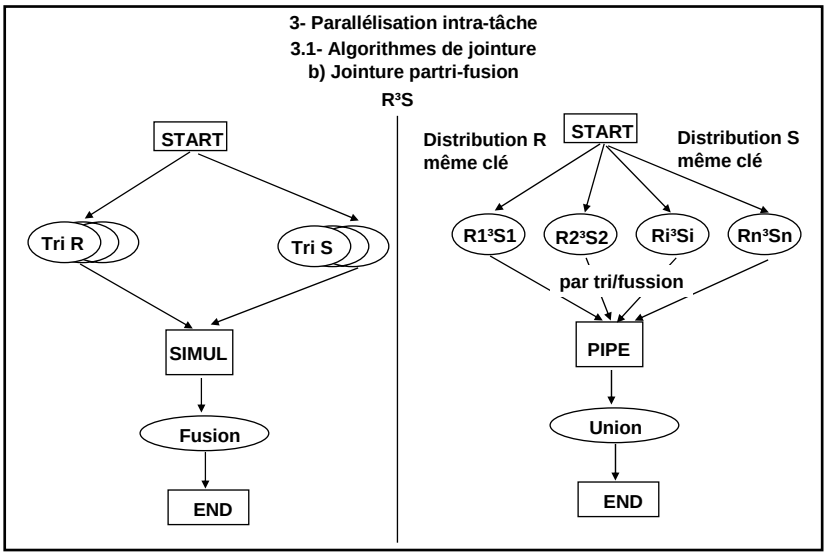
II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.1- Algorithmes de jointure

b) Jointure partri-fusion

R^3S



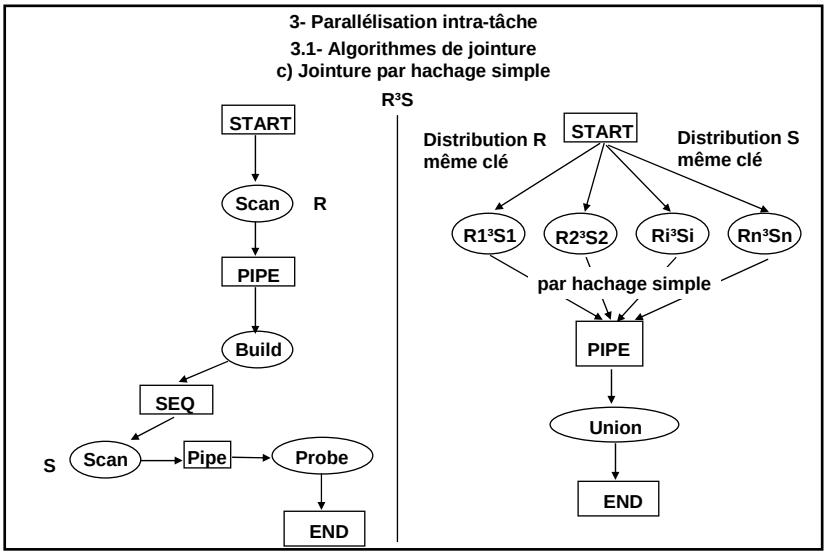
II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.1- Algorithmes de jointure

c) Jointure par hachage simple

R^3S

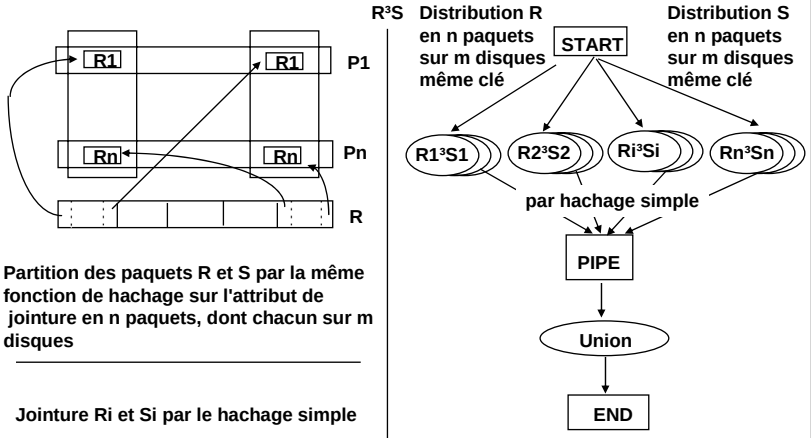


II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.1- Algorithmes de jointure

c) Jointure par Grace hash-join



Partition des paquets R et S par la même fonction de hachage sur l'attribut de jointure en n paquets, dont chacun sur m disques

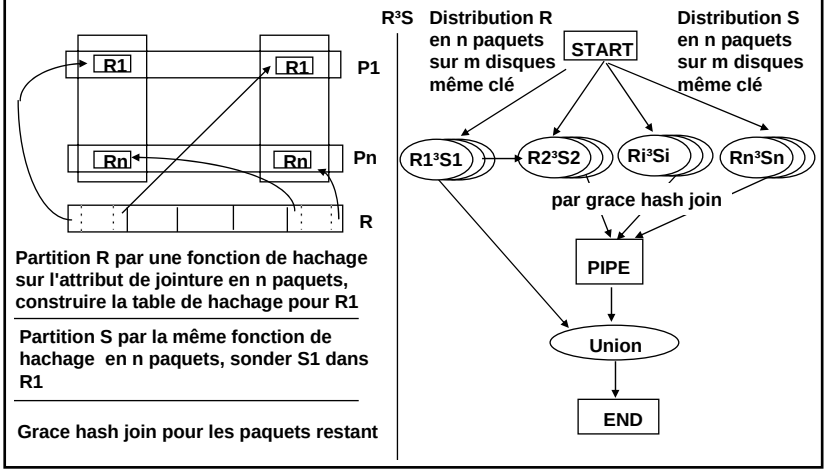
Jointure Ri et Si par le hachage simple

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.1- Algorithmes de jointure

c) Jointure par Hachage Hybide



Partition R par une fonction de hachage sur l'attribut de jointure en n paquets, construire la table de hachage pour R1

Partition S par la même fonction de hachage en n paquets, sonder S1 dans R1

Grace hash join pour les paquets restant

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

a) Notions

- **Règle de déduction : formule logique de la forme : P(X0) <-- Q1(X1), Q2(X2),...,Qn(Xn)**
  - **P est la tête de règle- Qi est un littéral du corps de la règle**
  - **Xi est un vecteur de variables - , est le symbole de conjonction logique**

- **Un prédicat P dépend directement de Q s'il existe une règle r où P apparaît dans l'entête de r et Q dans le corps de r**

- **P est récursif si P dépend de P**

- **P et Q sont multuellement récursifs ssi P dépend de Q et Q dépend de P**

- **Une requête récursive si elle fait référence à un prédicat récursif**

- **Quatre catégories de règles récursives :**
  - **linéaires (gauche et droite)** — **r1: A(x,y) <-- P(x,z), A(z,y)**
  - **fortement linéaires** — **r2: C(x,y) <-- P(xn,x1), C(x1,y1), Q(y,y1)**
  - **quadratiques** — **r3: A(x,y) <-- A(x,z), A(z,y)**
  - **multuellement récursives** — **r4: S(x1,z1) <-- M(x1,y1), T(y1,z1)**, **r5: T(y1,z1) <-- S(y1,w1), P(w1,z1)**

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

a) Notions

- **Un programme P est un ensemble de règles**

- **L'entrée I d'un programme est un ensemble fini des faits (tuples)**

- **La sortie O(I,P) d'un programme P respectant I est un ensemble fini de faits (tuples)**

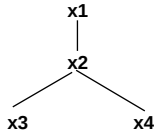
- **Fermeture transitive d'une relation: opérateur point fixe**

Soit R(X,Y) une relation binaire où X et Y sont définis sur le même somaine D. La relation R définie un graphe G dont les noeud sont des élément de D et les arcs (x,y) sont des tuples de R.

On appele fermeture transitive de R la relation R^+ contenant tous les tuples (x,y) tel qu'il existe un chemin allant de x à y dans G.

R

x1, x2
x2, x3
x2,x4



R^+

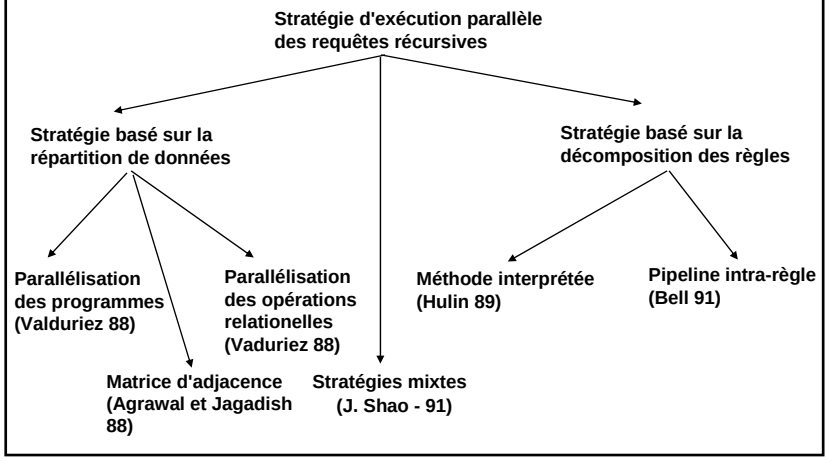
x1, x2
x2, x3
x2,x4
x1,x3
x1,x4

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

a) Notions



II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

b) Parallélisation des programmes

Algorithme de Fermeture Transitive Itérative (FTI)

r1: A(x,y) <-- P(x,y) ou r2: A(x,y) <-- A(x,z), P(z,y)

Calcul de la relation A

```
Procédure FTI(P: opérateur; A: Résultat)
début
        A:= P;
        D:= P;                          /* D contient les tuple de P */
        tanque D <> Ø faire
        début
                D := D ⊗ P;             /* ⊗ dénote l'opérateur de jointure)
                D := D - A; /* - dénote l'opérateur de différence */
                A := A ∪ D;             /* ∪ dénote l'opérateur d'union */
        fintanque
fin
```

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

b) Parallélisation des programmes

Algorithme de Fermeture Transitive de relation Fermées Transitivement (FTRFT) (Valduriez 88)

- Objectif : calcul de la fermeture transitive avec les programmes parallèles (même programme exécute sur différents processeurs)

- Principe : Soit R une relation binaire.
Si R est partitionnée en deux sous-ensembles R1 et R2, alors :

$$R^+ = (R1^+ \cup R2^+)^+$$

En utilisant l'algorithme FTI on a :

$$(R1^+ \cup R2^+) \otimes (R1^+ \cup R2^+) =$$
$$(R1 \overset{+}{\otimes} R1^+) \cup (R1^+ \otimes R2^+) \cup (R2^+ \otimes R1^+) \cup (R2 \overset{+}{\otimes} R2^+)$$

Or $(R1 \overset{+}{\otimes} R1^+) = R1^+$ **et** $(R2 \overset{+}{\otimes} R2^+) = R2^+$

Il suffit de caluler les deux opérandes $(R1^+ \otimes R2^+)$ **et** $(R2^+ \otimes R1^+)$

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

b) Parallélisation des programmes

Algorithme de Fermeture Transitive de relation Fermées Transitivement (FTRFT) (Valduriez 88)

```
Procédure FTRFT(R1 et R2 : opérande; T : résultat)
début
        flip := vrai;
        D1 := R1 ⊗ R2;
        D2 := R2 ⊗ R1;
        T := R1 ∪ R2 ∪ D1 ∪ D2;
        tanque D1 ° ∅ et D2 ° ∅
        début
                si flip alors D1 := D1 ⊗ R1 sinon D1 := D1 ⊗ R2;
                si flip alors D2 := D2 ⊗ R2 sinon D2 := D2 ⊗ R1;
                T := T ∪ D1 ∪ D2;
                flip := non flip;
        fintantque
fin
```

II- SGBDs PARALLELES

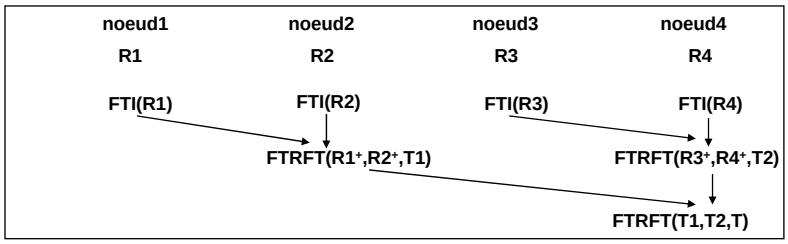
3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

b) Parallélisation des programmes

Algorithme de Fermeture Transitive avec les programmes parallèles (FTPP) (Valduriez 88)

- Nombre de processeurs $d=2^k$
- Principe :
  - R est répartie sur d processeurs
  - Phase 1 : FTI de chaque relation
  - phase 2 : FTRFT deux à deux
- Complexité : $O = \log_2 d$



II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

b) Parallélisation des programmes

Algorithme de Fermeture Transitive avec les programmes parallèles (FTPP) (Valduriez 88)

```
procédure FTPP (R : opérande, T : résultat)
début
        pour chaque noeud i (i = 1 ..d) faire
        début
                FTI(Ri,Ti);
                si i mod 2 ° 0 alors envoyer (Ti, noeud (i+1));
        finpour;
        pour j:= 1 jusqu'à [log2d] faire
                pour chaque noeud i (i = 2j-1, 2*2j-1, 3*2j-1,..., d) faire
                début
                        Sj := recevoir;
                        FTRFT (Sj,Si,Ti);
                        si i mod 2j ° 0 alors envoyer 'Ti, noeud (i+2j-1));
                finpour
        finpour
fin
```

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

b) Parallélisation des des opération relationnelles

Algorithme de Fermeture Transitive avec les opérations parallèles (FTOP) (Valduriez 88)

- Objectif : calculer la fermeture transitive en parallélisation des opérations relationnelle

- Principe :

- **dans l'algorithme FTI, la fermeture transitive est caculer par l'ittération de 2 opérations relationnelles : jointure et union.**
- **On peut donc envisager de parallélisation de ces deux opérations en partitionant la relation opérande en n sous-ensembles disjoints.**
- **L'algorithme s'exécute en deux phases :**

  - **phase 1 : distribuer la relation R(X,Y) sur d processeurs en utilisant une fonction de hachage appliquée sur l'attribut de jointure Y**

  - **phase 2 : caculer la fermeture transitive est calculée par une boucle qui distribue la relation D (résultat partiel initialisé avec R), avec la même fonction de hachage mais applquée cette fois sur l'attribut X, et effectue la jointure Di = Ri ⊗ Di.**

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

b) Parallélisation des des opération relationnelles

Algorithme de Fermeture Transitive avec les opérations parallèles (FTOP)

```
procédure FTOP (R:opérande, T:résultat)
début
        distribuer (R, h(Y));
        pour chaque noeud i (i:1..d) faire
        début
                Ti := Ri := recevoir;
                Di := Ri;
        fin pour
        tanque que (⋂ Di, i=1..d) ° ∅
        début
                distribuer (D,h(X));
                pour chaque noeud i (i=1..d) faire
                début
                        Di := recevour;
                        Di := Ri ⊗ Di;
                        Ti := Ti ∪ Di;
                fin pour
        fin tant que
fin
```

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

c) Fermeture Transitive s'appuyant sur la matrice d'adjacence

(Wharshall - Agrawal et Jagadish 88)

- Objectif : calculer la fermeture transitive d'une matrice booléenne représentant le graphe des élément d'une relation R

- Principe :

- **un graphe comportant n noeud est représenté par une matrices binaire d'adjacence n*n: s'il existe un arc entre le noeud i et le noeud j alors aij=1 sinon aij=0.**
- **L'algorithme conste à traiter tous les éléments de la matrice ligne par ligne de gauche à droite et de haut en bas.**
- **Chaque élément aij est traité de la manière suivante :**
  - **si aij = 1 alors tous les successeurs de j sont marqués sucesseurs de i**
- **Ce traitement doit respecter les deux contraintes suivantes :**
  - **C1 : Dans une ligne i, le traitement aik précède le traitement de aij si et seulement si k < j**
  - **C 2 : Pour tous les éléments aij dans la ligne i, le traitement de l'élement ajk précède le traitement de aij si et seulement si k < j**

- 41 -

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

c) Fermeture Transitive s'appuyant sur la matrice d'adjacence

(Wharshall - Agrawal et Jagadish 88)

- Principe de parallélisation:

- Phase 1 : partition et distribution de la matrices sur m processeurs
  - **Répartition de la relation R(X,Y) sur l'attribut X en donnant à chaque processeur un ensemble de lignes consécultives de la matrice d'adjacence**
  - **Les processeurs sont numérotés et le processeur p a pième partition de la matrice**

Phase 2 : Calcul parallèle de la fermeture transitive
  - **soit m processeurs (m partitions)**
  - **soit bp et ep la &ere ligne et la dernière ligne de la partition p sur le p ième processeur**
  - **le processeur p traite les éléments en dessous du diagonale de la bpet copier la ligne à tous autres processeurs qui récupérent cette ligne pour continuer de la traiter**

- 42 -

II- SGBDs PARALLELES

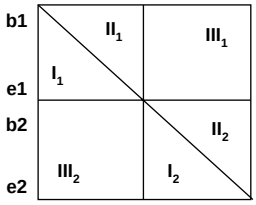
3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

c) Fermeture Transitive s'appuyant sur la matrice d'adjacence

(Wharshall - Agrawal et Jagadish 88)

- Principe de parallélisation:



- Le processeur 1 traite en premier les éléments de I_1

- Lors du calcul d'une ligne de cette région, dès que l'élément de la diagonale de cette région est atteint :
- **une copie de la ligne est faite et rendue disponible à tous les processeurs**
- **Le processeur 2 peut maintenant accéder à copie de cette ligne et l'utiliser pour calculer les élément de la région III_2. Il traite les éléments de la colonne ayant le numéro de la première ligne de la région**

- Entre temps, quand procésseur 1 termine de traiter la région I_1, il passe à traiter la région II_1 III_1
- Ainsi les élément de II_2 sont calculés en parallèle avec ceux de I_1
- Après le traitement des éléments de la région III_2, le processeur 2 traite les lignes de I_2 puis II_2

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.2- Parralélisation des requêtes récursives

c) Fermeture Transitive s'appuyant sur la matrice d'adjacence (Agrawal et Jagadish 88)

```
procédure FMA                          (Algorithme à exercuter sur un processeur p)
début                                  (avec m partitions pour m processeurs)
   pour q:=1 à  m faire
     si p=q alors        /*partition pour le processeur p*/
     début
            pour i:=bp à ep faire /*on traite les éléments au dessous de la diagonale*/
            début
                  pour j:=bp à  i-1faire traiter(aij);
                  copier (sucesseur(i), copie(i));
                  montrer(tous, copie(i));
            fin
            pour i:=bp à ep faire /*on traite les éléments au dessus de la diagonale*/
                  pour j:=i+1 à ep faire traiter (aij);
     fin
     sinon  pour j:=bq à eq faire
            début
                  récupérer(copie(j));
                  pour i:=bp à ep faire traiter(aij);
            fin
fin
```

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.3- Evaluation en pipeline des requêtes récursives (Bell 91)

a) Forme canonique des règles récursives

Toute requête peut être mise sous forme canonique :

R_i <-- A_{i1}, R_{i1}, A_i, R_{i2}, .., A_{in-1}, A_{in}

R_i <-- C_i

Q<-- R_1

où

- A_{ij} et C_i (1ŠiŠm, 1ŠjŠn) sont des ensembles (qui peuvent être vides) de conjonctions de prédicats de base

- R_1, ..., R_m sont des prédicats récursifs

- R_{ij} est un prédicat dans le corps d'une règle. Il corespond à un R_k prise à partir de l'ensemble des prédicats récursifs

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.3- Evaluation en pipeline des requêtes récursives (Bell 91)

b)Mise en forme normale et Ordonancement des prédicats

Deux phases de compilation :

- Etape 1 : Simplifier la requête par remplacer les prédicats non récursifs et déduits par sa définition et par le regroupement tous les prédicats de base en un seul

- Etape 2 : Ordonancement (heuristique "Making selection first")
pour chque prédicat pb dans une clause :

1. Si l'évaluation de pb permet de déduire des variables liées pour les variables de Rij, alors pb appartient à Aij

2. Si l'évaluation de Rij permet de déduire des variables liées pour les variables de pb, alors pb appartient à Aij

3. Si l'évaluation de pb permet de déduire une partie de la réponse R, alors pb appartient à Ci

4. Si p1 et p2 sont deux prédicats de base dans Aij ou Ci, p1 est évalué avant p2 (p1 est à la gauche de p2) si l'évaluation de p1 produit des variables liées pour p2

II- SGBDs PARALLELES

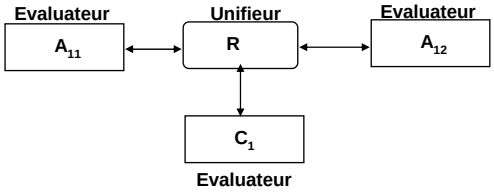
3- Parallélisation intra-tâche

3.3- Evaluation en pipeline des requêtes récursives (Bell 91)

c) Evaluation des requêtes en forme normale par pipeline

Soit une requête récursive linéaire

R <-- A_{11} R A_{12}
R <-- C_1
Q <-- R



- Evaluateur :
exécution un ensemble d'opérations relationnelles sur l'ensemble des relations de base qu'il possède et envoi le résutat E(O) à l'Unifieur

- Unifieur :
Unification des résultats reçus des éluateurs et génération du résutat final de la requête

- 47 -

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.3- Evaluation en pipeline des requêtes récursives (Bell 91)

c) Evaluation des requêtes en forme normale par pipeline

Evaluateur :

- Chaque évaluateur possède un ensemble de prédicats de base de la forme :

E(O) = b(I), p_1,p_2,..,p_k

où :

- I est l'ensemble des variables liées reçu initialement par l'unifieur
- O contient l'ensemble des valeurs calculées à partir de b(I), p_1,p_2,..,p_k ces valeurs sont envoyées à l'unifieur

- Principe de calcul :

1. La conjontion de deux prédicats est calculée par la jointure des relations de base correspondantes

2. Un prédicat lié partiellement est calculé par la selection de la relation de base correspondante sur la constante liée de ce prédicat

3. E(O) est calculé par la projection sur O de la rélation déduite du calcul de b(I), p_1,p_2,..,p_k

- 48 -

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.3- Evaluation en pipeline des requêtes récursives (Bell 91)

c) Evaluation des requêtes en forme normale par pipeline

Unifieur :

- L'unifieur possède 4 relations de travail :

- New_Instance et Old_Instance qui stockent les tuples provenant de A11

- New_Answer et Old_Answerqui ont le même schéma relationnel que R, sont utilisés pour stocker les tuple dérivés los du processus de calcul de la requête Ces relations sont des sur-ensembles du résultat.

- Un "marking scheme" = < M1, M2, ..., Mm>
(om m est l'arité des relation instance) est adssocié aux relations Instances. Il indique la correspondance entre les valeurs d'un tuple d'une relation Instance et les positions des arguments dans R.

Chaque Mi est un entier non négatif. Mi=j signifie que la valeur dans le ième argument d'une relation Instance doit être liées avec jième argument de la relation R

II- SGBDs PARALLELES

3- Parallélisation intra-tâche

3.3- Evaluation en pipeline des requêtes récursives (Bell 91)

c) Evaluation des requêtes en forme normale par pipeline

Unifieur : Principe de calcul

- Initialisation :
New_Instance := {constantes données dans la requête};
New_Answer := ;
Old_Instance := ;
Old_Answer := ;

- Etape 1 : Calcul de nouveaux tuples

itération jusqu'à VRAI de la Condition d'Arrêt
Temp_Instance := New_Instance - Old_Instance;

Temp_Answer := New_Answer - Old_Answer;
(Résultat de Temp_Instance est envoyé à A11 et C1)

- Etape 2 : Calcul des conjonctions

Join(Temp_Instance, Old_Answer);
Join(Old_Instance, Temp_Answer);
Join(Temps_Instance, Temp_Answer);

Résultat est envoyé à A12.

- Etape 3 : Calcul des nouvelles valeurs :

Old_Instance := New_Instance ∪ Old_Instance;
Old_Answer := New_Answer ∪ Old_Answer;
New_Instance := receive(A11);
New_Answer := receive(A12, C1);

- Etape 4 : Tester la condition d'arrêt
Si aucun tuple n'a été déduit par les évaluateurs alors sélection finale et arrêt
Sinon reitérer à partir l'étape 1.

II- SGBDs PARALLELES

3- Parallélisation intra-tâche
3.3- Evaluation en pipeline des requêtes récursives (Bell 91)
c) Evaluation des requêtes en forme normale par pipeline
Requêtes déductives non-linéaires

Soit une requête récursive non- linéaire (sous forme canonique stable)

$R \leftarrow A_{11}, R, A_{12}\ R, A_{13}\ R, ..., R, A_{1n}$
$R \leftarrow C_1$
$Q \leftarrow R$

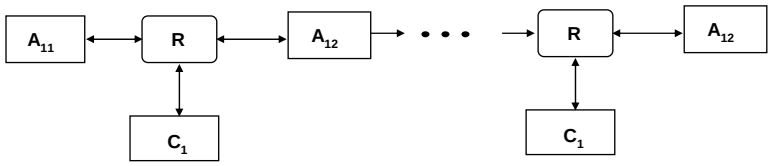


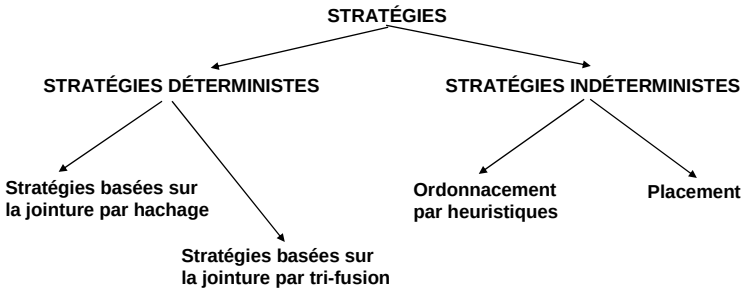
Schéma d'évaluation en pipeline d'un requête récursive non-linéaire

II- SGBDs PARALLELES

4- Parallélisation inter-tâche
4.1- Objectifs et principes

Objectifs :

ORDONNANCEMENT PARALLELE DES OPÉRATIONS RELATIONNELLES D'UNE MÊME TRANSACTION



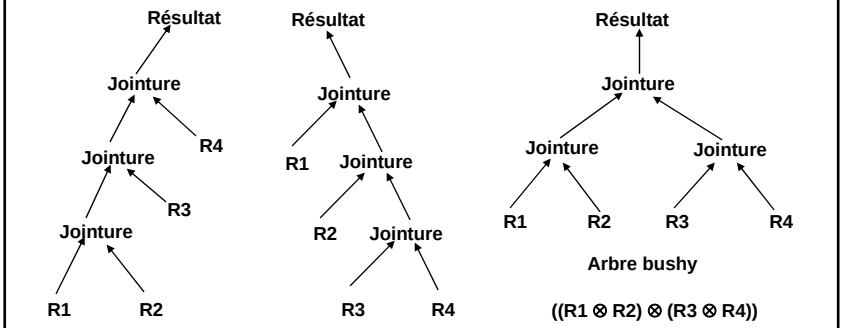
II- SGBDs PARALLELES

4- Parallélisation inter-tâche

4.2- Stratégies déterministes

1) Modélisation de l'espace de recherche

L'espace de recherche est modélisé par une arbre de jointure



Arbre linéaire gauche

(((R1 ⊗ R2)⊗ R3)⊗ R4)

Arbre linéaire droite

(R1 ⊗ (R2⊗ (R3 ⊗ R4)))

Arbre bushy

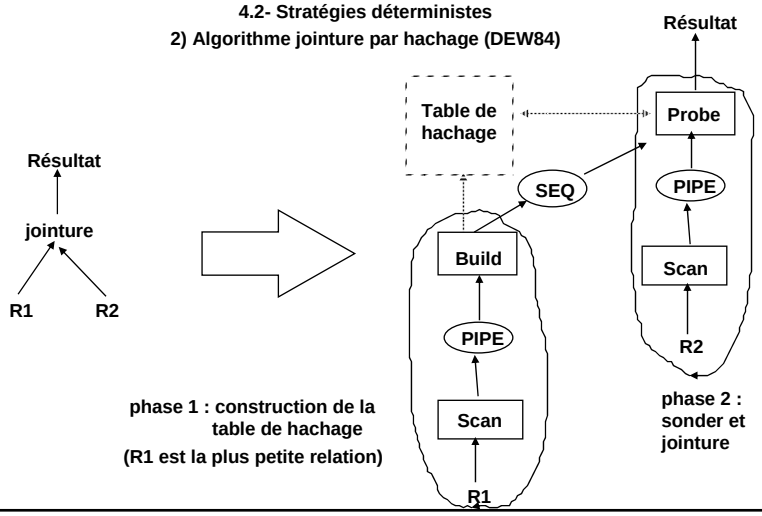
((R1 ⊗ R2) ⊗ (R3 ⊗ R4))

II- SGBDs PARALLELES

4- Parallélisation inter-tâche

4.2- Stratégies déterministes

2) Algorithme jointure par hachage (DEW84)



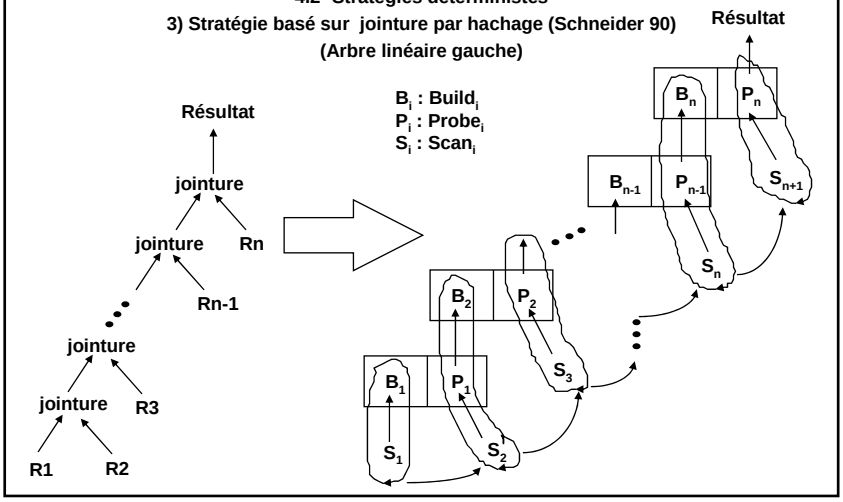
phase 1 : construction de la table de hachage

(R1 est la plus petite relation)

phase 2 : sonder et jointure

II- SGBDs PARALLELES

4- Parallélisation inter-tâche
4.2- Stratégies déterministes
3) Stratégie basé sur jointure par hachage (Schneider 90)
(Arbre linéaire gauche)



II- SGBDs PARALLELES

4- Parallélisation inter-tâche
4.2- Stratégies déterministes
3) Stratégie basé sur jointure par hachage (Schneider 90)
(Arbre linéaire gauche)

Seq

Pipe Scan1 - Build1 End_Pipe ;

Pipe Scan2 - Probe1 - Build2 End_Pipe;

• • •

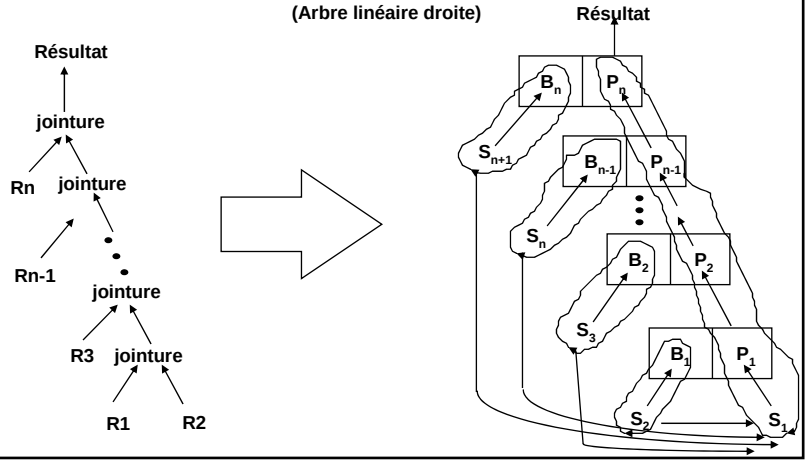
Pipe Scan$_n$ - Probe$_{n-1}$ - Build$_n$ End_Pipe;

Pipe Scan$_{n+1}$ - Probe$_n$ End_Pipe;

End_Seq

II- SGBDs PARALLELES

4- Parallélisation inter-tâche
4.2- Stratégies déterministes
3) Stratégie basé sur jointure par hachage (Schneider 90)
(Arbre linéaire droite)



II- SGBDs PARALLELES

4- Parallélisation inter-tâche
4.2- Stratégies déterministes
3) Stratégie basé sur jointure par hachage (Schneider 90)
(Arbre linéaire droite)

```
Seq
        Par
                Pipe Scan2 - Build1 End_Pipe ;

                Pipe Scan3 - Build2 End_Pipe;
                        • • •

                Pipe Scann+1 - Buildn End_Pipe;

        End_Par

        Pipe Scan1 - Probe1 - Probe2 ... Proben End_Pipe;

End_Seq
```

II- SGBDs PARALLELES

4- Parallélisation inter-tâche
4.2- Stratégies déterministes
3) Stratégie basé sur jointure par hachage (Schneider 90)
(Arbre linéaire droite : optmisation de mémoire)
Static right-deep scheduling

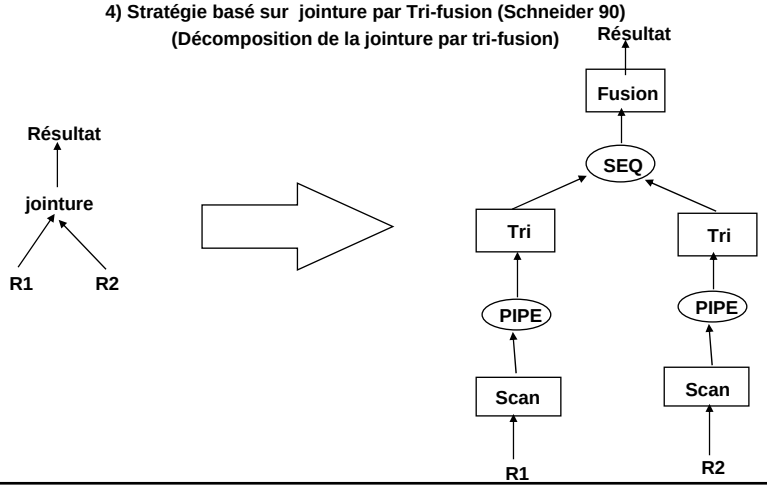
```
Seq
        Par
                Pipe Scan2 - Build1 End_Pipe ;
                Pipe Scan3 - Build2 End_Pipe;
        End_Par
        Pipe Scan1 - Probe1 - Probe2 End_Pipe;

        Par
                Pipe Scan4 - Build3 End_Pipe ;
                Pipe Scan5 - Build4 End_Pipe;
        End_Par
        Pipe Probe3 - Probe4 End_Pipe;
End_Seq
```

- 59 -

II- SGBDs PARALLELES

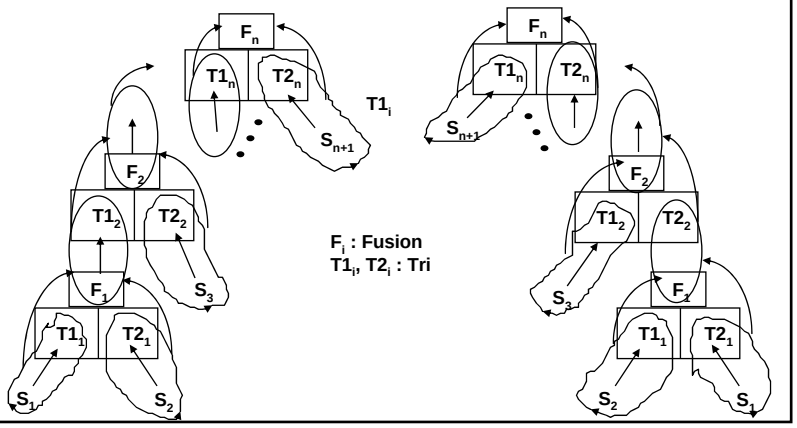
4- Parallélisation inter-tâche
4.2- Stratégies déterministes
4) Stratégie basé sur jointure par Tri-fusion (Schneider 90)
(Décomposition de la jointure par tri-fusion)



- 60 -

II- SGBDs PARALLELES

4- Parallélisation inter-tâche
4.2- Stratégies déterministes
4) Stratégie basé sur jointure par Tri-fusion (Schneider 90)
(Arbre linéaire gauche ou droite)



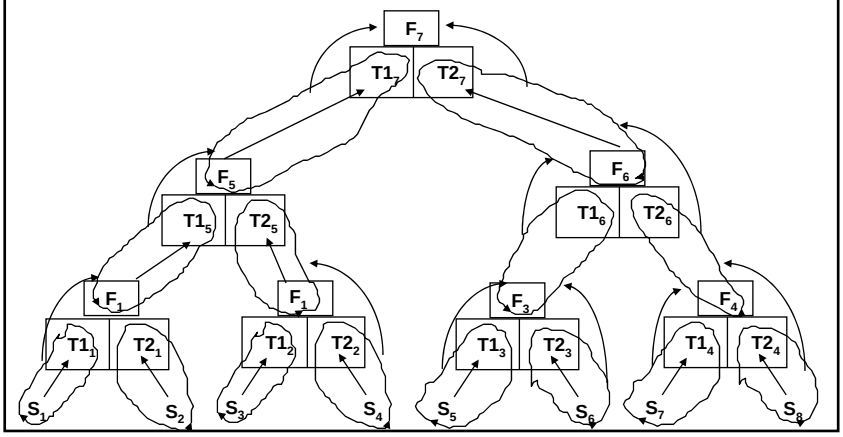
II- SGBDs PARALLELES

4- Parallélisation inter-tâche
4.2- Stratégies déterministes
4) Stratégie basé sur jointure par Tri-fusion (Schneider 90)
(Arbre linéaire gauche)

```
Seq
        Par
                Pipe Scan1 - Tri11 End_Pipe ;
                Pipe Scan2 - Tri21 End_Pipe;
        End_Par
        Par
                Pipe Fusion1 - Tri12 End_Pipe ;
                Pipe Scan3 - Tri22 End_Pipe;
        End_Par
                • • •
        Par
                Pipe Fusionn-1 - Tri1n End_Pipe ;
                Pipe Scann+1 - Tri2n End_Pipe;
        End_Par
        Fusionn;
End_Seq
```

II- SGBDs PARALLELES

4- Parallélisation inter-tâche
4.2- Stratégies déterministes
4) Stratégie basé sur jointure par Tri-fusion (Schneider 90)
(Arbre Bushy)



II- SGBDs PARALLELES

4- Parallélisation inter-tâche
4.2- Stratégies déterministes
4) Stratégie basé sur jointure par Tri-fusion (Schneider 90)
(Arbre Bushy)

```
Seq

 Par
        Pipe Scan1 - Tri11 End_Pipe ;
        Pipe Scan2 - Tri21 End_Pipe;
        Pipe Scan3 - Tri12 End_Pipe;
        Pipe Scan4 - Tri22 End_Pipe;
        Pipe Scan5 - Tri13 End_Pipe;
        Pipe Scan6 - Tri23 End_Pipe;
        Pipe Scan7 - Tri14 End_Pipe;
        Pipe Scan8 - Tri24 End_Pipe;
 End_Par;

 Par
        Pipe Fusion1 - Tri15 End_Pipe ;
        Pipe Fusion2  - Tri25 End_Pipe;
        Pipe Fusion3 - Tri16 End_Pipe ;
        Pipe Fusion4 - Tri26 End_Pipe;
 End_Par;

 Par
        Pipe Fusion5 - Tri17 End_Pipe ;
        Pipe Fusion6  - Tri27 End_Pipe;
 End_Par;
  Fusion8;

End_Seq
```

II- SGBDs PARALLELES

4- Parallélisation inter-tâche

4.3- Stratégies indéterministes

1) Ordonnancement - un problème NP-Complexe

• Rappels :

- **Un problème P0 appartient à NP s'il est accepté par une machine Turing non déterminste en un temps polynomial**
- **Un problème P0 est NP-difficile si tout problème de NP peut être réduit polynomialement en P0 (le nombre de sous-problème P0 qu'il faut pour résoudre P est borné par un polynôme en fonction de la taille des paramètres de P)**
- **Un problème P0 est dit NP-complet s'il appartient à NP et est NP-difficile**

• Exemples des problèmes NP-complexe

- **Voyage de commerce**
- **Circuit hamiltonien**

• Théorème (Ph. Chrétienne - 88) : Le problème d'ordonnancement des tâches sur un nombre fini de processeurs, en général, est un problème NP-complexe

II- SGBDs PARALLELES

4- Parallélisation inter-tâche

4.3- Stratégies indéterministes

1) Ordonnancement - un problème NP-Complexe

Deux propriétés fondamentales de la classe de NP-complexe

- **Aucun problème NP-complexe ne peut être résolu par un algorithme de complexité polynomiale**
- **S'il existe un algorithme de complexité polynomiale pour résoudre un problème NP-complexe, alors il en existe pour chaque problème NP-complexe**

Conséquence :

Il existe pas d'algorithme polynomial comme solution aux problèmes NP-complexes

Solution :

Utilisation des algorithmes qui cherche des solutions approchées en un temps résonnable (polynomial)

II- SGBDs PARALLELES

4- Parallélisation inter-tâche

4.3- Stratégies indéterministes

2) Classification des méthodes

• Méthode de recherche de la solution exacte :

- algorithmes recherche la meilleur en exploitant toutes les solutions possibles

- Exemple : Trouver toutes les solutions pour placerr n tâches indépendantes sur 2 processeurs

- On trouve : 2^{n-1} de solutions

- Avec un processeur Alpha de 300 Mips et supposons qu'il peut évaluer 100 millions solutions différentes par seconde

- En sachant qu'il y a $3,16 \times 10^9$ secondes dans un siècle

- Il faut donc un siècle pour évaluer $3,16 \times 10^{17}$ solution

- Or $259 = 5,76 \times 10^{17}$. Il faut donc près 2 siècles pour évaluer toutes les solutions de placer 60 tâches sur 2 processeurs !!!

II- SGBDs PARALLELES

4- Parallélisation inter-tâche

4.3- Stratégies indéterministes

2) Classification des Méthodes

Méthodes déterministes :

Emposer une méthode déterministe pour rechercher un ordonnancement raisonnable

Méthodes itératives :

- Recherche d'une solution satisfaisante en transformant itérativement une solution initiale dans une bonne direction
(problème : Miinimums locaux à éviter)

- Exemples : le récuit simulé, la recherche tabou, les algorithmes génétiques

Méthodes de liste :

- basée sur la composition de certaines heuristiques on définit un ordre totale entre les tâches. On ordonnance des tâches par la stratégie glouton : l'exécution d'une tâche s'effectue dès que possible, dans l'ordre de la liste et sans attente, si les contraintes de précédence la permettre

II- SGBDs PARALLELES

4- Parallélisation inter-tâche
4.3- Stratégies indéterministes
3) Principe des algorithmes de liste

- **Les heuristiques**

»**LP (*Longest Path*) : Donne la priorité à la tâche origine du plus long chemin.**
»**longueur du chemin fait référence aux temps d'exécution des tâches. La**
»**longueur d'une tâche s'obtient par un calcul récursif**

»**LSF (*Least Schedule Flexibility*) : Donne la priorité à la tâche dont la date**
»**d'exécution ne peut être repousée sans allonger le temps de réponse.**

»**LPT (*Longest Processing Time*) : Donne la priorité à la tâche qui possède la**
»**durée d'exécution la plus longue.**

»**HL (*Highest Level*) : Donne la priorité à la tâche qui est de niveau le plus**
»**haut dans le graphe de tâches. Le niveau 0 est celui des tâches terminales.**

II- SGBDs PARALLELES

4- Parallélisation inter-tâche
4.3- Stratégies indéterministes
3) Principe des algorithmes de liste

- **Principe**

»**1. Trier les tâches selon les heuristiques**
»**2. Dans l'ordre défini à l'étape 1 :**
-**Chercher une place pour la première tâche de la liste ¼ Ddeb1, Dfin1.**
-**Si Ddeb1 > Dtôt alors chercher, en réduisant le degré de parallélisme de la tâche, Ddeb2 telle que : Dfin2 < Dfin1.**
-**Prévoir la tâche pour qu'elle se termine au plus tôt.**
»**3. Passer à la tâche suivante.**

- **Ajout des contraintes de placement**

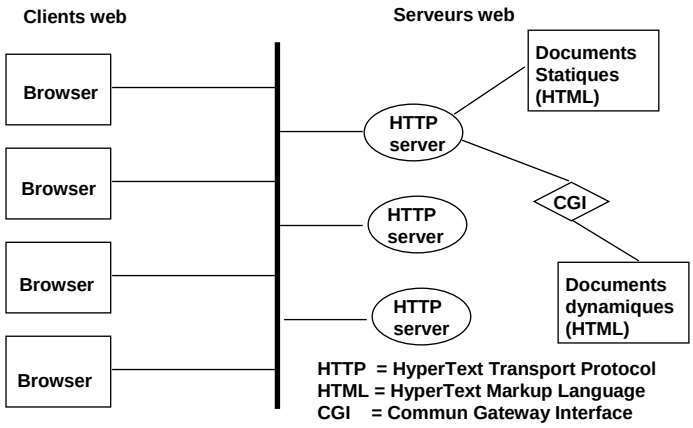
- **Ajout des communications**

III- ORAWEB

- Environnement de web
- Programmation client/serveur avec le SGBD ORACLE
- Prinicipes et architechture de ORAWEB
- Extension de PLSQL pour web
- Perspectives : programmation dynamique avec les SGBDs

III- ORAWEB

1- Environnement de web
1.1- Principaux composants



III- ORAWEB

1- Environnement de web

1.2- Protocole HTTP

Principe d'une transaction

- Une transaction de travail consiste en

- une connexion	**établissement de la connexion par un client au serveur en utilisant le port TCP/IP n° 80 (par défaut) ou un port cité dans URL (Uniform Ressource Locator)**
- une requête	**le client envoie une requête au serveur**
- une réponse	**le serveur envoie une réponse au client**
- une fermeture	**la fermeture de la connexion peut se faire par une des deux parties**

- Le protocole HTTP ne gére pas la couche session

- HTTP est un demon qui travaille sur le port 80 par défaut. On peut utiliser un autre port à préciser au lancement de HTTP

III- ORAWEB

1- Environnement de web

1.2- Protocole HTTP

Utilisation et configuration

- Lancement :

httpd [options] &

Options :

- HTTP est un demon qui travaille sur le port 80 par défaut. On peut utiliser un autre port à préciser au lancement de HTTP

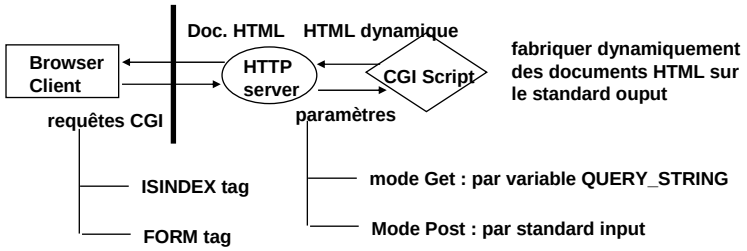
III- ORAWEB

1- Environnement de web

1.2- Protocole HTTP

Interface CGI (Common Gateway Interface)

- Technique standard utilisée par le serveur HTTP pour exécuter un programme qui génère des sorties HTML (HTML dynamique)



Format de requête CGI : chemin_virtuel_chemin_extra?paramètres
Format des paramètres : nomvar1=val1&nomvar2=val2&...&nomvaln=valn
Format des valeurs texte : " "remplacé par "+", caractère = %codeASCCI

III- ORAWEB

1- Environnement de web

1.3- Langage HTML

- Langage standard permettant de définir les documents hypermédias sur web

- Quatre fonctions :

- **structuration**
- **présentation**
- **cheminement**
- **saisie et activation des traitements**

- Les documents sont constitués par des "HTML tags" :

- **tags d'entête : informations générales (facultatives)**
- **tags du corp: le contenu du document (obligatoire)**

- format général d'un tag :

```
<nom tag>          (début d'un tag)
informations
</nom tag>         (find'un  tag)
```

III- ORAWEB

1- Environnement de web

1.3- Langage HTML

Structure du document

```
<HTML>
   <HEAD>
      <TITLE>  Titre du document </TITLE>
   </HEAD>
   <BODY>
      <H1>
            <H2>
                  ...
                        <H6>
                        .....
                        </H6>
            </H2>
      </H1>
   </BODY>
</HTML>
```


III- ORAWEB

1- Environnement de web

1.3- Langage HTML

Les Basic HTML tags

Ouverture	Fermeture	Définition
<HTML>	</HTML>	Document en HTML
<HEAD>	</HEAD>	Entête du document
<TITLE>	</TITLE>	Titre du document
<BODY>	</BODY>	Contenu du document
<H1>	</H1>	Entête du 1er niveau
<H2>	</H2>	Entête du 2e niveau
<H3>	</H3>	Entête du 3e niveau
<H4>	</H4>	Entête du 4e niveau
<H5>	</H5>	Entête du 5e niveau
<H6>	</H6>	Entête du 6e niveau
<P>		Terminaison d'un paragraphe
<PRE>	</PRE>	Texte pré-formatté
 		Saut de ligne
<BLOCKQUOTE>	</BLOCKQUOTE>	Texte cité à partir d'une autre souces

III- ORAWEB

1- Environnement de web

1.3- Langage HTML

Les List tags

- Liste ordonée : Les rubriques sont numérotées

<OL> {<LI>} </OL>

- Liste non ordonée : Les rubriques sont préfixées par le symbole •

<UL> {<LI>} </UL>

- Liste avec définition : des termes avec leur définition

<DL> {<DT>/<DD>} </DL>

Exemple :

```
<DL>
<DT> HTML :
<DD> HyperText Markup Language
<DT> HTTP :
<DD> HyperText Transport Protocol
</DL>
```


III- ORAWEB

1- Environnement de web

1.3- Langage HTML

Les tags de style

- texte gras	**<B> texte </B>**
- texte italic	**<I> texte </I>**
- texte souligné	**<U> texte </U>**
- texte monospacé (typewriter)	**<TT> texte </TT>**

Les accents et caractères spéciaux

- Macro & :

- accents :

&egrave	**è**
&agrave	**à**
...	

- Les caractères réservés :

&lt	**<**
&gt	**>**
&amp	**&**
&quot	**"**

Tag ADDRESS

```
<ADDRESS>
adresse_de_l'auteur
</ADDRESS>
```

III- ORAWEB

1- Environnement de web

1.3- Langage HTML

Les liens d'Hypertexte

URL (Uniform Resouce Locators) : Pointeur sur réseaux internet

méthod://machine-name:port/path/file#ancre

- *method* protocole d'accès : file (local) - http- ftp - mailto (sans //) ...

- *machine-name:port* nom de la machine et n° de port sur lequel un serveur web est lancé

- *path/file#ancre* le chemin complet du fichier avec un point d'accès particulier (ancre) facultatif

ANCRE: tag de définition d'un pointeur dans un document

<A HREF="URL"> nom_ancre_suraffiché </A>

III- ORAWEB

1- Environnement de web

1.3- Langage HTML

Les liens à l'intérieur d'un document

- Création des marqueurs (référence muette) pour marquer la première ligne des sections à référencer :

<A NAME="nom_du_marqueur"> Texte_début_de_la_section </A>

- Création des liens vers ces marqueurs :

< A HREF="#nom_du_marqueur"> nom_lien </A>

Exemple :

- marqueur :

<H2> <A NAME="intro"> Introduction aux SGBDs </A> </H2>

- lien :

<A HREF="#intro"> Les SGBDs </A>

III- ORAWEB

1- Environnement de web

1.3- Langage HTML

Les tags graphiques

```
<IMG  SRC="URL"
  ALT="text"
  ALIGN={"top"  | "middle" | "bottom" | "texttop"}
  ISMAP >
```

- **SRC** poiteur réseau vers le fichier d'image (.gif ou .jpeg)*
- **ALT** le texte à afficher si le browser est incapable d'afficher l'image
- **ALIGN** le placement du bloc de texte qui suit l'image, par défaut le texte est placé à droite de l'image par le browser
- **ISMAP** image bitmap

* Note : certains browsers acceptent les formats MPEG et QUICK TIME

III- ORAWEB

1- Environnement de web

1.3- Langage HTML

Les tags basics pour les tableaux

```
<TABLE  [BORDER]>  LES_DONNEES </TABLE>

<TR [ALIGNE={left | center | right}] [VALIGNE={top | middle | bottom | baseline}]>
</TR>

<TH       [ALIGNE={left | center | right}]
          [VALIGNE={top | middle | bottom | baseline}]
          [COLSPAN =nb_de_colonnes]
          [ROWSPAN =nb_de_lignes]
          [NOWRAP]>
</TH>

<TD       [ALIGNE={left | center | right}]
          [VALIGNE={top | middle | bottom | baseline}]
          [COLSPAN =nb_de_colonnes]
          [ROWSPAN =nb_de_lignes]
          [NOWRAP]>
</TD>
```

```
<CAPTION
  [ALIGNE={top | bottom}]
le_tittre
</CAPTION>
```

III- ORAWEB

1- Environnement de web

1.3- Langage HTML

Les tags basics pour les tableaux

```
<TABLE BORDER> <CAPTION ALIGN=bottom> Table n°1 </CAPTION>
<TR> <TD ROWSPAN=2> </TD> <TH COLSPAN=2> Average </TH> </TR>
<TR> <TH>Height</TH> <TH> Weight </TH> </TR>
<TR> <TD>Males</TD> <TD ALIGN=center> 69 </TD>
<TD ALIGN=center> 150 </TD></TR>
<TR> <TD>Females</TD> <TD ALIGN=center> 64 </TD>
<TD ALIGN=center> 130 </TD></TR>
```

	Average	
	Height	Weight
Males	69	150
Females	64	130

III- ORAWEB

1- Environnement de web

1.3- Langage HTML

Les tags basics pour les forms

- Les forms sont utilisés pour la saisie des données et l'activation des traitements adéquats sur le serveur depuis un poste client

```
<FORM METHOD="get | post"
ACTION="URL">éléments_de_form
</FORM>
```

- METHOD : La méthode de transmettre les données au programme
- **get : l'information est transmises à la fin de la demande de l'action URL. Le programme recoit le données par l'intermédia de la variable d'environnement QUERY_STRIING**
- **post : l'information est transfert avec l'action URL au programme. La variable d'environnement CONTENT_LENGTH stocke le nombre d'octets de données à lire sur le standard input**

- ACTION : indique l'URL demandée par le formulaire. Cet URL pointe sur un CGI script qui organise le traitement des données venant du formulaire

III- ORAWEB

1- Environnement de web

1.3- Langage HTML

Les tags basics pour les forms : zones de saisie

Zone de saisie de texte ASCCI TEXTAREA tag

```
<TEXTAREA
    NAME= "nom_de_la_zone"
    ROWS= largueur en caractères de TEXTAREA
    COLS= longueur en caractères de TEXTAREA>
Texte_defaut
</TEXTAREA>
```

Saisie des caracètres (une ligne maximum) : INPUT tag

```
<INPUT
 NAME= "nom_de_la_zone_de_saisie"
 CHECKED mode de saisie "checkbox" ou "bouton radio"
 MAXLENGTH= largueur max en caractères de la ligne (défaut 20)
SIZE= largueur de la fenêtre de saisie (peut-être < MAXLENGTH)
SRC= URL spécifiant une image (si TYPE=IMAGE)
TYPE= CHECKBOX / HIDDEN / IMAGE / PASSWORD / RADIO / RESET / SUBMIT / TEXT
VALUE= valeur_par_defaut
</INPUT>
```


III- ORAWEB

1- Environnement de web

1.3- Langage HTML

Les tags basics pour les forms : Menus de sélection

```
<SELECT |SELECT SINGLE | SELECT MULTIPLE
 NAME= nom de la variable>
<OPTION> Texte_option1
<OPTION> Texte_option 2
...
</SELECT>
```

SELECT tag : afficher 1ère option par défaut - bouton d'afficher les autres options, une seule peut-être sélectionnée

SELECT SINGLE tag : les options sont affichées dans une fenêtre avec scroll bar, une seule peut être sélectionnée (3 options sont affichiées par défaut)

SELECT SINGLE tag : les options sont affichées dans une fenêtre avec scroll bar, mais on peut selectionner plusieurs options *

*** dans certains browsers, il faut taper CTRL + SHIFT pour une sélection multiple**

III- ORAWEB

1- Environnement de web

1.4- Langage JAVA

Introduction

- Origine : 1er langage de programmation pour web - SUN - version bêta

- Principe d'exécution : langage interprété par une machine virtuel BYTECODE

- adapter à toutes les plate-formes matérielles (indépandance matérielle)

- renforcer la sécurité et la performance

- Principes de programmation : à objets similaire à C++ avec quelques caractéristiques

- pas de pointeur : moins de supplesse mais plus propre et sécurisé

- muti-tâche (multi-thread) : pour adapter à l'environnement multi-processeurs

- gestion de mémoire intégrée : mécanisme de ramasse-miettes intégré (garbage collector) permettant de décharger le programmeur de la tâche de gestion de la mémoire

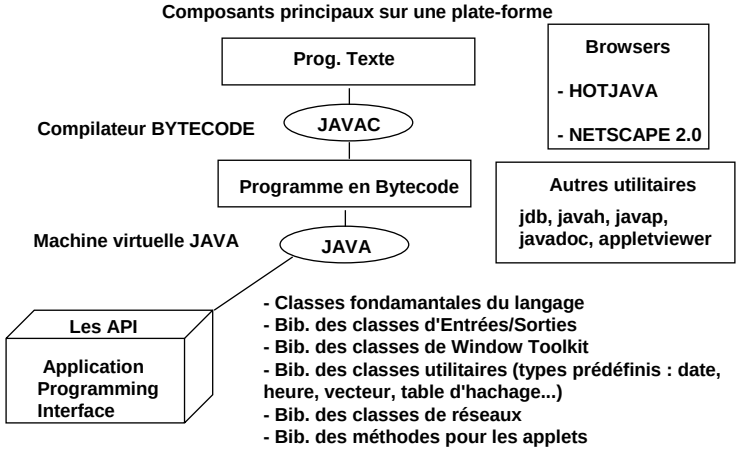
- mode de programmation avec HTML (APPLET)

III- ORAWEB

1- Environnement de web

1.4- Langage JAVA

Composants principaux sur une plate-forme



III- ORAWEB

1- Environnement de web

1.4- Langage JAVA

Utilisation

- Compilation : **javac [options] nom_du_prog_source .java**

Options :

classpath path	**instanciation de la var. CLASSPATH**
-d répertoire	**répertoire stockant la classe compilée**
-g	**exécution pas à pas (n° de ligne et des variables)**
-o	**optimisation**
-v | -verbose	**afficher les messages de compilation**

- CLASSPATH : **variable d'environnement qui stocke des chemins d'accès aux classes utilisées par le programme**

Syntaxe : **path1:path2:...:pathn**

III- ORAWEB

1- Environnement de web

1.4- Langage JAVA

Utilisation

Exécutiond'une classe prinicpale (la classe contenant la méthode main()) (interprete de pseudo-code) :

java [options] nom_de_classe .class <args>

Options :

-debug	**exécution pas à pas avec jdb**
-checksource | cs	**vérifier la version de source, recompiler si nécessaire**
classpath path	**instanciation de la var. CLASSPATH**
-mx x	**chager la taille max de l'allocation de mémoire à x (du ramasse-miette - défaut 16 Mo**
-ms x	**changer la taille du départ du ramasse-miettes par défaut 1Mo**
-noasyncgc	**désartiver le ramasse-miettes asynchrone. Il sera appelé explicitement ou lorsque la mémoire allouée au prog. n'est pas suffisante**
-ss x ou -oss x	**modofier la taille de la pile de code C(ss) ou java (oss)**
-t	**tracer les instructions exécutées**
-v | -verbose	**affichier un message à chaque chargement d'une classe**
-DpropertyName=v	**redéfinir la valeur d'une propriété Ex. : java -Dawt.button.color=green**

III- ORAWEB

1- Environnement de web

1.4- Langage JAVA

Utilisation

Debugger de Java : jdb

Méthode 1 : Utilisation locale

jdb nom_de_classe .class <args>

jdb -classpath path nom_de_classe .class <args>

Méthode 2 : Utilisation à distance sur réseaux

jdb -host <hostname> -password <password>

-host <hostname>	**nom de la machine hôte sur laquelle l'interpréteur est exécuté**
-password <password>	**connexion sur la session de l'interpréteur actif**

Commandes :	**help | ?**	**affichier la liste de commades**
	print	**décrire les classes, on spécifie la classe par son n° ID ou par son nom (EX: print java.lan.Thread)**
	dump	**décrire le contenu des variables de la classe spécifiée**
	threads	**lister les tâche courantes**
	where	**décrire le contenu de la pile de la tâche spécifiée**

Spécifier les breackpoints : stop at nom_de_classe:numéro_de_ligne

III- ORAWEB

1- Environnement de web

1.4- Langage JAVA

Utilisation

Générateur de source en C: javah

Utilisation :

générer les sources en C pour implanter les méthodes "natives" en C d'une classe java

Syntaxe :

javah [option] nom_de_classe .class

Options :

-o fichier_de_sortie	**concaténer le fichier généré dans fichier_de_sortie**
-d répertoire	**répertoire où stocke le fichier généré**
-td répertoire	**changer le répertoire par défaut (tmp) de sauvegarde**
-verbose	**afficher les statuts des fichiers générés**
-classpath path	**spécifier le chemin où javah ira chercher les classes**

III- ORAWEB

1- Environnement de web
1.4- Langage JAVA
Utilisation

Déassemblteur des fichier compilé par java : javap

Utilisation :
déassembler les fichiers compilés par java et afficher une représentation du pseudo-code

Syntaxe :
javap [options] nom_de_classe .class

Options :

-l	**afficher les variables locales et les n° de ligne**
-p	**afficher les méthodes et les champs privés et protégés**
-c	**afficher le code désasemblage**
-classpath path	**spécifier le chemin où javah ira chercher les classes**

III- ORAWEB

1- Environnement de web
1.4- Langage JAVA
Utilisation

Générateur de document d'API en format HTML : javadoc

Utilisation :
analyser les déclarations et les commentaires des fichiers sources en java et créer des documents d'explication en format HTML. Javadoc tient compte des classes, interfaces, méthodes et variables

Syntaxe :
javadoc [options] package | nom_de_fichier.java

Options :

-d répertoire	**spécifier le répertoire où javadoc va stocker les documents en HTML**
-classpath path	**spécifier le chemin où javah ira chercher les classes**
-verbose	**affichier les informations concernant les fichiers sources (compilés ?, modifiés ? ...)**

III- ORAWEB

1- Environnement de web
1.4- Langage JAVA
Utilisation

Survol de la syntaxe du langage java

- Les commentaires :

// texte	**une ligne de commentaire**
/* texte */	**une ou plusieurs lignes de commentaires**
/ texte */**	**le commentaire sera analysé par javadoc**

- Les identificateurs : lettres miniscules ou majuscules, chiffres

- Les mots réservés :

abstract	**boolean**	**break**	**byte**	**byvalue**	**case**	**catch**
char	**class**	**const**	**continue**	**default**	**do**	**double**
else	**extends**	**false**	**final**	**finally**	**float**	**for**
goto	**if**	**implements**		**import**	**instanceof**	**int**
interface	**long**	**native**	**new**	**null**	**package**	**private**
protected	**public**	**return**	**short**	**static**	**super**	**switch**
synchronized	**this**	**threadsafe**		**throw**	**transient**	**true**
try	**void**	**while**				

- 97 -

III- ORAWEB

1- Environnement de web
1.4- Langage JAVA
Utilisation

Survol de la syntaxe du langage java

- Les littéraux :
- **les entiers : quatre classes : byte, short, int et long**
- **les flottants : deux classes : float et double**
- **les booléens : deux valeurs : true et false**
- **les caractères : de classe char**
- **les chaînes de caractères : classe String (° tableau de cars en C)**
- **les tableux : création dynamique par new: (ex. char s[] = new char[20]), pas de tableaux multidimentionnels, l'index doit-être entier, la longueur est calculée par la méthode length**

- La conversion entre classes : (nom_de_classe) ref (ref est objet à convertir)

- les classes :

[commentaires_javadoc] [spécificateurs] class Nom_de_Classe
[extends Nom_de_superClase]
[implements Interfaces] {corps_de_la_classe}

Spécificateurs = abstract | final

- 98 -

III- ORAWEB

1- Environnement de web
1.4- Langage JAVA
Utilisation

Survol de la syntaxe du langage java

- Les constructeurs : méthodes d'initialisation des objets
- **dans une classe A : même nom de la classe : A(paramettres)**
- **pour les objets de la super_classe : super(paramettres)**
- **ils sont appelés automatiquement à la création d'un objet**

- **les caractères : de classe char**
- **les chaînes de caractères : classe String (° tableau de cars en C)**
- **les tableux : création dynamique par new: (ex. char s[] = new char[20]), pas de tableaux multidimentionnels, l'index doit-être entier, la longueur est calculée par la méthode length**

- Les méthodes :
[Commentaires_javadoc] [Spécificateurs]
Type_de_retour Nom_méthode(paramettres)
{corps_de_la_méthode}
Spécificateurs = public | private | protected | threadsafe | native \ synchronized

III- ORAWEB

1- Environnement de web
1.4- Langage JAVA
Utilisation

Survol de la syntaxe du langage java

- Les structures :
if **(boolean) déclarations** ***else*** **déclarations;**
switch **(e1) {** ***case*** **e2 : déclarations**
...
defaut : déclarations};
break **[label] :**
continue **[label] ;**
return **e1;**
for **([e1] ; [e2] ; [e3]) déclarations** ***while*** **(boolean) déclarations;**
do **déclarations** ***while*** **(boolean);**
label **déclarations;**

- **les tableux : création dynamique par new: (ex. char s[] = new char[20]), pas de tableaux multidimentionnels, l'index doit-être entier, la longueur est calculée par la méthode length**

- Les exceptions : ***throw*** **Nom_de_l'Objet** **lever une exection**

III- ORAWEB

1- Environnement de web
1.4- Langage JAVA
Utilisation

Survol de la syntaxe du langage java

- Les applets :

- Les classes java qui doient être intégrées dans un document HTML

- Les applets héritent la super_classe java.applet.Applet

- Une classe qui définit d'un applet peut être surchargée quatre méthodes de la classe java.applet.Applet :
- **init() qui l'initialise à chaque chergement**
- **start() qui lance l'exécution de l'applet**
- **stop() qui définit son arrêt lorsque l'utilisateur quite la page contenant l'applet ou quite le browser**
- **destroy() qui nettoye la page de l'applet**

- Tag applet dans un document HTML :

```
<APPLET  CODE=Nom_de_classe.class
         WIDTH= largeur de la fenêtre en caractères
         HEIGTH= hauteur de la fenêtre en caractères>
</APPLET>
```

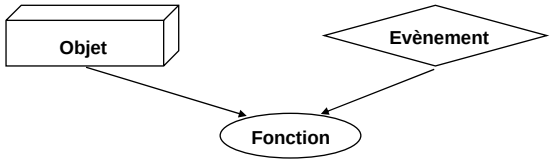

III- ORAWEB

1- Environnement de web
1.5- Le langage JavaScript
Introduction

- langage script basé sur java et utilisé dans un document HTML (Netscape+Sun)

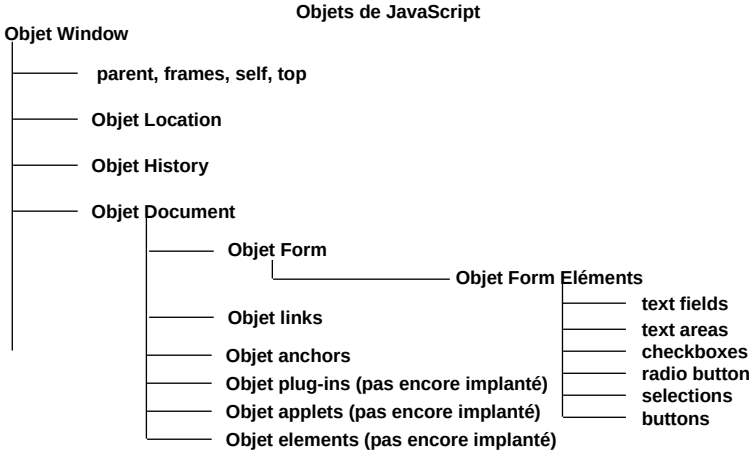
- le JavaScript est ressenble à java mais avec le typage dynamique, il supporte la plupart des expressions de Java

- le JavaScript
- **connaît les objets d'un document HTML**
- **gére un ensemble des évènements sur ces objets**
- **gére un ensemble de fonctions, écrirtes en javascript ou applet java, et attachées à un document**
- **peut activer des fonctions quand un évenèment s'introduit sur un objet**



III- ORAWEB

1- Environnement de web
1.5- Le langage JavaScript
Objets de JavaScript



III- ORAWEB

1- Environnement de web
1.5- Le langage JavaScript
Les évènements

- Events sont des actions qui s'introduisent comme le résultat une action de l'utilisateur (Exemple : "click" est un évènement sur un bouton)

- **Focus, Blur, Change : text fields, textareas, selections**
- **Click : buttons, radi button groups, check boxes, submit buttons, reset buttons**
- **Select : textfields, textareas**

- Event handlers sont des scripts qui exécutent automatiquement quand un évènement s'introduit (utilisés usuellement avec les forms et les éléments de form)

- onFocur= "JavaScript fonction"	**curseur est sur l'élément de form**
- onBlur= "JavaScript fonction"	**curseur est quitté l'élément de form**
- onChange= "JavaScript fonction"	**valeur de l'élémént de form est changée**
- onSelect= "JavaScript fonction"	**option séléctionnée**
- onSubmit= "JavaScript fonction"	**form est submit**
- onClickr= "JavaScript fonction"	**form est reset**

```
<INPUT TYPE="button" VALUE="Calulateur" onClick="compute(this.form)">
```

III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Les objets pré-définis de JavaScript

- Window : objet racine de chaque document HTML

- propriétés :

- frames[index]	**tableau de frame wildows fils (dans l'ordre défini)**
- frames.length	**nombre de frames fils définis**
- self	**le window courant**
- parent	**le window père**
- top	**le window origin**

- Méthodes

- alert("string")	**afficher la fenêtre d'alert avec le message**
- confirm("string")	**afficher la fenêtre de confimation avec le message (avec OK / CANCEL)**
- open("URL", "name")	**ouvrir une nouvelle fenêtre client avec nom, charger l'URL dans cette fenêtre**
- close()	**close la fenêtre**

III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Les objets pré-définis de JavaScript

- Location : contenant les information sur l'URL courent

- propriétés :

- href	**l'URL entier comme un JavaScript String**
- protocol	**la sous-chaine de protocole (y comprise le ":")**
- host	**la partie hostname:port**
- hostname	**le hostname**
- port	**le port**
- path	**le chemin**
- hash	**le chemin CGI après #**
- search	**le chemin CGI après ?**
- post	**(pas encore implanté)**

- Méthodes

- tostring()	**returner location.href**
- assign()	**charger location.href**

III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Les objets pré-définis de JavaScript

- History : contenant les information sur les URLs étant visités par le client. Les informations sont stockées dans un history list et accessible par menu Go

- propriétés :

- back	**URL précedent**
- forward	**URL suivant**
-current	**URL de la page courante**
- length	**Longueur de la liste de historique**

- Méthodes

- go(delta)	**delta est un nombre entier. Si delta > 0 charger l' URL courent + delta, si < 0, charger l' URLcourant - delta**
- go("string")	**charger 'URL cité**
- toString()	**retourner une chaine contenant un tableau HTML de tous les liens dans la liste de historique**

III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Les objets pré-définis de JavaScript

- Document : contenant les information sur ledocument en cours

- propriétés :

- title	**titre du document**
- location	**URL complet du document**
- lastmodified	**date de la dernière modification (string)**
- loadedDate	**pas encore implanté**
- referer	**pas encore implanté**
- bgcolor	**valeur RGB de la couleur de fond (background)**
- fgcolor	**valeur RGB de la couleur de texte (foreground)**
- linkcolor	**valeur RGB de la couleur des hyperlinks**
- vlinkcolor	**valeur RGB de la couleur des hyperlinks visités**
- alinkcolor	**valeur RGB de la couleur des liens actifs (boutons)**
- forms[index]	**tableau des objets form dans l'ordre d'apparition**
- forms.length	**nombre des objets form dans le documents**
- links[index]	**tableau des HREF dans l'ordre d'apprition**
- links.length	**nombre de liens dans le documents**
- anchors[index]	**tableau des ancres (anchor tag) dans l'ordre**
- anchors.length	**nombre des ancres dans le document**

III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Les objets pré-définis de JavaScript

- Document : (suite)

- Méthodes :

- write(arguments)	**afficher dans le document HTML des arguments comme un texte HTML**
- writeln(arguments)	**sauter la ligne et afficher dans le document HTML des arguments comme un texte HTML**
-clear()	**effacer la fenêtre**
- open("MIME type")	**pas encore implanté**
- close()	**pas encore implanté**

- Exemple

```
document.write("<PRE>")
document.writeln ("Nom de rubrique :" + document.forms[0].num.name)
document.writeln ("Valeur de rubrique :" + document.forms[0].num.value)
document.write("</PRE>")
```


III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Les objets pré-définis de JavaScript

- Forms : Chaque form est un objet distinct. Il existe un tableau des objets forms dans chaque document. Les forms sont référencés par leur numéro dans ce tableau

- Propriétés :

- name	**Valeur du l'attribut NAME**
- method	**valeur de l'attribut METHOD (get ou post (0/1)**
- action	**valeur de l'attribut ACTION**
- target	**fenêtre de réponse**

- Event Handlers

- onSubmit="fonction de JavaScript à exécuter"

- Méthodes

- submit()	**soumettre le form**

III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Les objets pré-définis de JavaScript

- Text Elements : text et textarea partagent les mêmes propriétés et méthodes

- Propriétés :

- name	**Valeur du l'attribut NAME**
- value	**valeur du contenu de la zone**
- defaultValue	**valeur par défaut de la zone**

- Event Handlers

- **onFocus="fonction de JavaScript à exécuter"**
- **onBlur="fonction de JavaScript à exécuter"**
- **onSelect="fonction de JavaScript à exécuter"**
- **onChange="fonction de JavaScript à exécuter"**

- Méthodes

- focus()	**focaliser l'entrée de données sur l'objet**
- blur()	**quitter l'entrée de données de l'objet**
- select()	**sélectionner la valeur de l'objet**

III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Les objets pré-définis de JavaScript

- Checkboxes : pour saisie d'une valeur booléenne ("on" ou "off")

- Propriétés :

- name	**Valeur du l'attribut NAME**
- value	**valeur du contenu de la zone : "on" ou "off"**
-status	**boolée, false si not checked true si checked**
- defaultStatus	**booléen qui indique si élément est sélectionné par défaut par l'attribut CHECKED**

- Event Handlers

- **onClick="fonction de JavaScript à exécuter"**

- Méthodes

- click()	**sélectionner le checkboxe s'il est sur "on"**

- Exemple :

```
<INPUT TYPE="checkboxe" NAME="checkoption1"
CHECKED onClick="update(this.form)">
OptionText
```

III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Les objets pré-définis de JavaScript

- Radio buttons : pour saisie d'une valeur parmi quelques unes
  - Propriétés :
    - name — Valeur du l'attribut NAME
    - value — valeur del'attribut VALUE
    - index — Numéro de l'option (de 0)
    - status — booléen, false si not checked true si checked
    - defaultStatus — booléen qui indique si élément est sélectionné par défaut par l'attribut CHECKED
  - Event Handlers
    - onClick="fonction de JavaScript à exécuter"
  - Méthodes
    - click() — sélectionner un radio bouton
  - Exemple :

```
<INPUT TYPE="radio" NAME="choix" VALUE="Cher"
onClick="catalog.value = "haut-de-gamme"> C'est une grande qualité
<INPUT TYPE="radio" NAME="choix" VALUE="Modest"
onClick="catalog.value = "milieu de gamme"> C'est une bonne qualité
<INPUT TYPE="radio" NAME="choix" VALUE="bon marché"
onClick="catalog.value = "bas-de-gamme"> C'est une mauvaise qualité
```


III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Les objets pré-définis de JavaScript

- Menu de sélection : pour saisie d'une option d'un menu
  - Propriétés de chaque option :
    - text — text de l'option
    - value — valeur de l'attribut VALUE à envoyer au serveur
    - index — Numéro de l'option dans le menu(de 0)
    - selected — booléen, false si not selected true si selected
    - defaultSelected — booléen qui indique si élément est sélectionné par défaut par l'attribut SELECTED
  - Event Handlers
    - onFocus="fonction de JavaScript à exécuter"
    - onFlur="fonction de JavaScript à exécuter"
    - onChange="fonction de JavaScript à exécuter"
  - Méthodes
    - click() — sélectionner une option
  - Exemple :

```
<SELECT NAME="select1" onChange="update(this.form)"
<OPTION> rouge
<OPTION> verte
<OPTION SELECTED> bleue
</SELECT>
```

III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Les objets pré-définis de JavaScript

- Boutons : trois types de boutons : bouton submit - bouton reset - bouton custom

- Propriétés de chaque option :
- **- name** valeur de l'attribut NAME
- **- value** valeur de l'attribut VALUE

- Event Handlers
- **- onClick="fonction de JavaScript à exécuter"**

- Méthodes
- **- click()** sélectionner le bouton

- Exemple :

```
<INPUT TYPE="submit" NAME="submit Buttom" VALUE="Submit Query">

<INPUT TYPE="reset" NAME="Reset Buttom" VALUE="Clear Form">

<INPUT TYPE="button" NAME="calculButtom" VALUE="Calculateur"
onClick="ma_fonction(this.form)">
```


III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Objets, propriétés, tableaux et méthodes de JavaScript

- Approche de l'objet singléton : chaque objet a ses propriétés et ses méthodes

- 3 manières de représentation des propriétés d'un objet :

Structure	Tableau indexé	Tableau ordinaire
voiture.marque = "Renault"	voiture[marque] = "Renault"	voiture[0] = "Renault"
voiture.model="R19"	voiture[model]="R19"	voiture[1]="R19"
voiture.année=95	voiture[année]=95	voiture[2]=95

- Les méthodes sont des fonctions normales du JavaScript avec un attachement à un objet :

objet.nom_méthode = nom_fonction

- On peut donc, appeller cette fonction par le nom de la méthode :

objet.nom_méthode(paramètres)

III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Syntaxe du langage

- Noms : suite des lettres des caractères "_" et des chiffres qui commence par une lettre ou la caractère "_"

- Litéraux : 4 types : entier - point flottant - booléen - chaîne de caractères

- entier : - base 10 : suite des chiffres dont le 1er ° 0
- base 8 : suite des chiffres dont le 1er = 0
- base 16 : suite des chiffres dont les 2 premiers = 0x

- flottant : - entier décimale: 1
- point décimale : 1.2223
- fraction
- exponentiel : 1.1 e 12 -2.13 E 11
- type surfixe

- booléen : deux valeurs : TRUE et FALSE

- chaîne de caractères : les caractères enveloppés par deux doubles quotes (") ou deux simples quotes un quote (')

III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Syntaxe du langage

- Expressions : un ensemble valable des litéraux, variables, opérations expressions qui peut être évaluée en une valeur numérique, textuelle ou logique

- Opérateurs d'assignement : (=, +=, -=, *=, /=, %=, <<=, >>=, >>>=, &=, ^=, |=)

x += y x = x+y

- Opérateurs :

- arthrimétiques :	**+, -, *, /, % (modulo)**
- logiques sur des bits :	**AND &, OR |, XOR ¨, << >> et >>> (scanne)**
- logiques :	**&& (et), || (ou), ! (non)**
- comparaison :	**==, >, >=, <, <=, !=**
- Incrément :	**++ (++x ou x++)**
- Décrément :	**-- (--x ou x--)**
- négation :	**- (-x)**
- chaînes :	**+ (concarténation)**

III- ORAWEB

1- Environnement de web
1.5- Le langage JavaScript
Syntaxe du langage
Instructions

- Mots réservés utilisés :

break	falsef	if	return	with
continue	for	in	this	while
else	function	null	true	var

- Mots réservés pour l'utilisation future :

abstract	const	float	native
default	goto	new	switch
implements	package	synchronized	byvalue
private	threadsafe	case	double
throw	catch	extends	int
char	final	interface	short
class	finally	long	static
super	byte	do	instanceof
public	try	void	boolean
delete	import	protected	transient

III- ORAWEB

1- Environnement de web
1.5- Le langage JavaScript
Syntaxe du langage
Instructions

- Break : terminaison d'une boucle

- Commentaires : // texte ou /* texte */

- Continue : terminaison d'une itération dans une boucle, recommence une nouvelle

- For Loop :

for ([initiale_expression;] [condition;] [update_expression]) { statements }

Initiale_expression = statement | variable_declaration

- Initiale_expression : initialisation de la variable de compteur
- Condition : condition à évaluer au début de chaque itération
- update_expression : mises à jour de la variable de compteur à la fin de chaque itération

III- ORAWEB

1- Environnement de web
1.5- Le langage JavaScript
Syntaxe du langage
Instructions

- FOR .. IN

    for (var in obj) {statements}

    var sont des index des propriétés de l'objet obj

- FUNCTION

    function_name([para] [,para] [..., para]) { statements]

- IL ... ELSE

    if (condition) {statements} [else { statements}]

RETURN

        return expression

III- ORAWEB

1- Environnement de web
1.5- Le langage JavaScript
Syntaxe du langage
Instructions

- THIS : objet courant

    This.propriété | méthode

- VAR : définition d'une variable

    var nom_var [= valeur] [..., var nom_var [= valeur]]

- WHILE

    while (condition) {statements}

- WITH : utilisation avec un objet pour éviter la répétition de son nom

    with objet {statements]

III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Utilisation dans le document HTML

- Script tag

```
<!-- debut de la partie invisible pour les anciens browses
      <SCRIPT
      LANGUAGE="langage"          langage : LiveScript, Java, ...
      SRC="URL">                  fichier de programme (.ls pour JavaScript)
      [Script text]
      </SCRIPT>
// fin de la partie invisible
```

- Exemple :

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="LiveScript">
document.write("Hello net.")
</SCRIPT>
</HEAD>
<BODY> That's all, folks. </BODY>
</HTML>
```


III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Utilisation dans le document HTML

- EXEMPLE 2 : une fonction simple

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="LiveScript">
<!-- debut de la partie invisible pour les anciens browses
      function carre(i)
            {
                  document.write("Vous donnez "), i, " a la fonction.", "<BR>")
                  return i * i
            }
            document.write("Lafonction retourne : "), square(5), ".")
// fin de la partie invisible
</SCRIPT>
</HEAD>
<BODY> <BR> Tout fait. </BODY>
</HTML>
```

III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Utilisation dans le document HTML

- EXEMPLE 3 : un script avec form simple

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="LiveScript">
        functioncacul(form) {if confirm("etes-vous sur ?"))
                                form.resut.value = eval(form.expr.value)
                            else alert("refaire s'il vous plait")
                            }
</SCRIPT>
</HEAD>
<BODY>
<FORM>
Entrer une expression :
<INPUT TYPE="text" NAME="expr" SIZE=30>
<INPUT TYPE="button" VALUE="Calculateur" ONCLICK="calcul(this.form)"> <BR>
Resultat :
<INPUT TYPE="text" NAME="result" SIZE=25> <BR>
</FORM> </BODY> </HTML>
```


III- ORAWEB

1- Environnement de web

1.5- Le langage JavaScript

Utilisation dans le document HTML

- EXEMPLE 4 : un script avec form

```
<HTML> <HEAD> <SCRIPT LANGUAGE="LiveScript">
        function testenum(str, min max)
                {if str=="") { alert ("Entrer un nombre s'il vous plait.")
                        return false }
                for (var i = 0; i < str.length; i++) {
                        var ch = str.substring(i, i+1)
                        if (ch < "0" || ch > "0") { alert("rentrer un numéro s'il vous plait")
                                                    return false            }           }
                return true }
        function merci() { alert("Merci pour votre test") }
</SCRIPT> </HEAD>
<BODY> <FORM>
Entrer une nombre entre 1 et 1000 :
<INPUT NAME="num" SIZE=15 ONCHANGE="if (testenum(this.value, 1, 1000))
                        { this.focus(); this.select();} else {merci()} VALUE="0"> </FORM>
<SCRIPT LANGUAGE="LiveScript"> document.write("<PRE>")
document.writeln("Nom de rubrique : " +  document.form[0]num.name)
document.writeln("Valeur de rubrique : " +  document.form[0]num.value)
document.write("<PRE>") </SCRIPT> </BODY> </HTML>
```

III- ORAWEB

2- Environnement de oraweb
2.1- Principe et architecture