Analyse théorique des algorithmes de jointure:

I. Jointure par Produit Cartésien:

On suppose que l'on a deux relations: R(X,Y), S(Y,X), et que l'on applique la jointure suivante: T=Jointure(R,S,R.Y=S.Y)

Fonctionnement:

1-lire la page i de R

2-lire la page j de S

3-effectuer les 4 matches possibles: R.Pi.Y0 ?= S.Pj.Y0

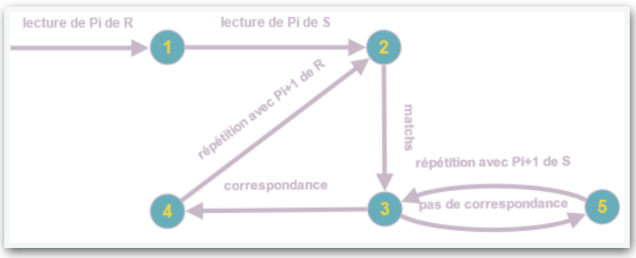
R.Pi.Y1 ?= S.Pj.Y0

R.Pi.Y1 ?= S.Pj.Y1

R.Pi.Y0 ?= S.Pj.Y1

4-s'il y a une correspondance alors l'écrire dans le résultat de la jointure (par exemple T.append(R.Pi.X0, R.Pi.Y0, S.Pi.Y0, R.Pi.X0)) et répéter les étapes 2,3,4,5 avec Pi+1 de R

5-si pas de correspondance alors lire Pj+1 de S et répéter les étapes 3,4,5



Graphe représentant le fonctionnement

de la jointure par Produit Cartésien

Avantage:

Convient pour des relations non triées et avec une distribution de données quelconque.

Inconvénient :

La durée d'exécution est généralement élevée, car elle nécessite de parcourir toutes les combinaisons possibles de tuples des relations.

Nombre de lectures et écritures disques:

-taille(R) lectures pour R

-taille(R)*taille(S) lectures pour S

-min(taille(R),taille(S)) écritures pour T

Total=taille(R)+taille(R)*taille(S)+min(taille(R),taille(S)) lectures/écritures disque.

II. Jointure par Tri-Fusion:

On suppose que l'on a deux relations déjà triées par ordre croissant en fonction de à leur Y: R'(X,Y), S'(Y,X), et que l'on applique la même jointure: T=Jointure(R',S',R'.Y=S'.Y)

Fonctionnement:

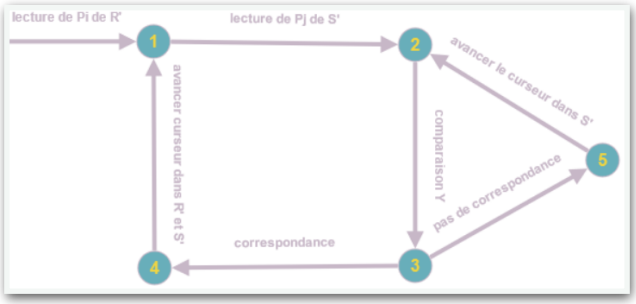
1-lire la page i de R'

2-lire la page j de S'

3- comparer R'.Pi.Y0 avec S.Pj.Y0 et R'.Pi.Y1 avec S.Pj.Y1

4-s'il y a une correspondance alors l'écrire dans le résultat de la jointure (T), avancer le curseur dans R' et dans S' et répéter les étapes 2,3,4,5.

5-s'il n y a pas de correspondance, si R'.Pi.Y0>S'.Pj.Y0 alors avancer le curseur dans S' (Y(1) ou passer à Pj+1 si déjà dans Y(1)) et répéter toutes les étapes.



Graphe représentant le fonctionnement

de la jointure par Tri-Fusion

Avantage:

Pas besoin de construire des index ou des tables de hachage.

Inconvénient :

Le tri initial peut être coûteux en termes de temps et de ressources dans le cas où il y a de grandes relations.

Nombre de lectures et écritures disques:

-taille(R) lectures pour R

-taille(S) lectures pour S

-min(taille(R),taille(S)) écritures pour T

Total=taille(R)+taille(S)+min(taille(R),taille(S)) lectures/écritures disque.

III. Jointure par Hachage Simple:

On suppose que l'on reprend les deux relations: R(X,Y), S(Y,X) utilisées pour le produit cartésien (par conséquent non triées). On suppose également que l'on a une table de hachage ayant autant d'entrée que de résultat possible de la fonction de hachage, et que l'on applique la même jointure sur Y:
T=Jointure(R,S,R.Y=S.Y)

Fonctionnement:

Phase 1: Build

On suppose que l'on a choisit la fonction de hachage modulo 3.

1-lire Pi de R

2-pour chaque coupe (X,Y) calculer le modulo 3 de Y et l'écrire dans l'indice de la table de hachage correspondant (hachage[Y%3].append(R.Pi.X0, R.Pi.Y0, S.Pi.Y0, R.Pi.X0)) en formant des blocs de deux éléments, répéter ces étapes avec R.Pi+1

Le résultat obtenu est une nouvelle version de la relation R ou les Y ont été regroupés par la fonction de hachage.

Phase 2: Probe

1-lire la page i de S

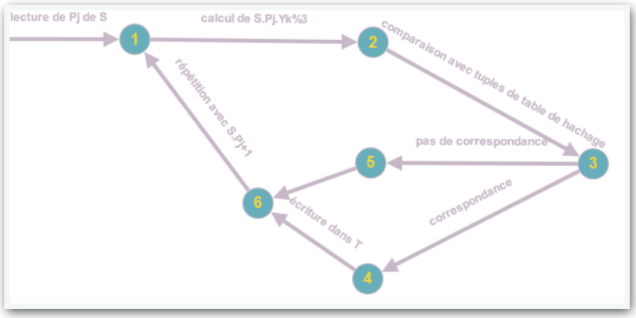
2-calculer S.Pj.Y0%3

3-chercher un tuple pour lequel Y vaut S.Pj.Y0%3 uniquement dans l'entrée S.Pj.Y0%3 de "hachage"

4-s'il y a une correspondance l'écrire dans T entrées

5-si pas de correspondance alors arrêter la recherche sans vérifier les autres entrées

6-reproduire les memes étapes avec S.Pj.Y1 puis avec S.Pj+1



Graphe représentant le fonctionnement

de la jointure par Hachage Simple

Avantages:

Utilise une table de hachage pour une recherche rapide des correspondances.
Peut gérer efficacement les données non triées.

Inconvénient :

Nécessite l'élaboration d'une table de hachage, ce qui peut être coûteux en termes de mémoire et de ressources lors de la phase de « Build ».

Nombre de lectures et écritures disques:

-taille(R) lectures pour R

-taille(S) lectures pour S

-min(taille(R),taille(S)) écritures pour T

Total=taille(R)+taille(S)+min(taille(R),taille(S)) lectures/écritures disque.

IV. Jointure par Produit Cartésien Indexé:

On suppose que l'on a deux relations: R(X,Y), S(Y,X), et que l'on applique la jointure suivante: T=Jointure(R,S,R.Y=S.Y)

Fonctionnement:

Il faut mettre en oeuvre un index I sur l'attribut Y de S. Cette structure stocke les valeurs de Y triées dans l'ordre croissant et permet d'accéder à chacune de manière efficace et rapide.

Pour chaque tuple dans R, il faut chercher les correspondances dans l'index et combine les tuples correspondants de R et S pour former le tuple résultant.

1-lire la page i de R

2-lire la valeur j de I

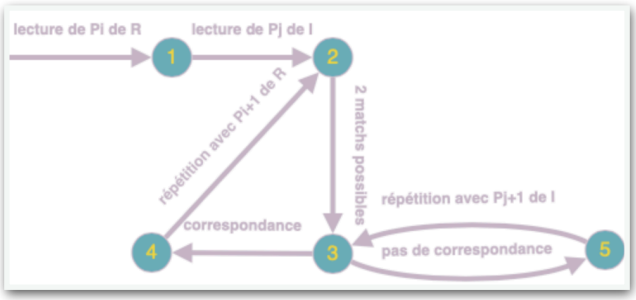
3-effectuer les 2 matches possibles: R.Pi.Y0 ?= I.Pj

R.Pi.Y1 ?= I.Pj

4-s'il y a une correspondance alors l'écrire dans le résultat de la jointure (T.append(R.Pi.X0, R.Pi.Y0,

S.(I.Pj).Y0, S.(I.Pj).X0) par exemple) et répéter les étapes 2,3,4,5 avec Pi+1 de R

5-si pas de correspondance alors lire Pj+1 de I et répéter les étapes 3,4,5



Graphe représentant le fonctionnement

de la jointure par Produit Cartésien Indexé

Avantages:

Utilise un index pour une recherche rapide des correspondances.
Peut gérer efficacement des relations non triées.

Inconvénient :

Nécessite l'élaboration d'un index, ce qui peut être coûteux en termes de mémoire et de ressources lors de la phase de prétraitement.

Nombre de lectures et écritures disques:

-taille(R) lectures pour R

-taille(S) lectures pour S

-min(taille(R),taille(S)) écritures pour T

Total=taille(R)+taille(S)+min(taille(R),taille(S)) lectures/écritures disque.

V. Jointure par Hachage Hybride:

On suppose que l'on reprend les deux relations: R(X,Y), S(Y,X) (non triées). On suppose également que l'on a une table de hachage ayant autant d'entrée que de résultat possible de la fonction de hachage, et que l'on applique la même jointure sur Y: T=Jointure(R,S,R.Y=S.Y).

M représentera ici la mémoire principale et F le facteur d'incrémentation.

Fonctionnement:

Il faut tout d'abord vérifier l'équation suivante: taille(R)*F <= taille(M).

Si elle est vrai cela signifierai qu'une table de hachage pour R tiendrai en mémoire réelle, dans ce cas le hachage hybride serait identique au hachage simple.

Dans le cas contraire il faudrait suivre les étapes ci-dessous:

-Calculer B = (taille(R)*F-taille(M))/taille(M)-1

-Partitionner R en R0,…,RB de façon à ce qu'une table de hachage pour R0 ait une taille de taille(M)—B blocs, et que R1, ..., Rb soient de même taille.

-Allouer ensuite B blocs (tampons) de sortie en mémoire. Attribuer les autres |M|-B blocs de mémoire à une table de hachage pour R0.

-Affecter le ième bloc tampon de sortie à Ri pour i=1,…,B

-Lire R et hacher chaque tuple avec la fonction de hachage. S'il appartient à R0 il sera placé en memoire dans une table de hachage. Sinon, le déplacer vers le ième bloc tampon de sortie.

-Partitionner S en S0,…,SB

-Affecter le ième bloc tampon de sortie à Si pour i=1,…,B

-Lire S et hacher chaque tuple avec la fonction de hachage. S'il appartient à S0 rechercher une correspondance dans la table de hachage en mémoire, en cas de production d'un résultat l'écrire dans T. Sinon, le déplacer vers le ième bloc tampon de sortie

Pour i=1,…,B:

-Lire Ri et construire une table de hachage en mémoire

-Lire Si, hacher chaque tuple et chercher une correspondance dans la table de hachage de R (en mémoire). Si un résultat est produit l'écrire dans T.

Avantages:
Utilise une combinaison de hachage en mémoire et d'accès disque pour minimiser les opérations de lecture et écriture disque, ce qui peut conduire à des gains de performance.
Peut s'adapter à différentes tailles de mémoire et de relations, ce qui le rend flexible dans des environnements avec des contraintes de mémoire.

Inconvénients:
L'efficacité de l'algorithme dépend de la capacité de la mémoire principale, si celle ci est trop petite il pourrait y avoir une augmentation des lectures/écritures disque.
La phase initiale de construction des tables de hachage peut nécessiter des ressources importantes, notamment en termes de temps et de mémoire.

Nombre de lectures et écritures disques:

-taille(R) lectures pour R

-taille(S) lectures pour S

-min(taille(R),taille(S)) écritures pour T

Total=taille(R)+taille(S)+min(taille(R),taille(S)) lectures/écritures disque.

IV. Comparaison:

Après avoir effectué une analyse théorique des différents algorithmes de jointures nous pouvons en conclure qu'en terme de lectures/écritures disques ce sont les algorithmes Tri-Fusion, Hachage simple et Produit Cartésien Indexé qui sont les plus efficaces.

En effet, leur exécution engendre le même nombre de lectures/écritures alors que le Produit Cartésien en engendre bien plus.

Néanmoins, certaines spécificités des ces trois algorithmes peuvent engendrer une distanciation par rapport aux autres. Ainsi, le Tri-Fusion semble être plus couteux et avoir un temps d'exécution plus long que les deux autres car il est nécessaire de trier les donner.

Le Hachage Simple et le Produit Cartésien Indexé nécessites chacun l'élaboration d'une table de hachage ou d'un index ce qui est également couteux.

L'analyse expérimentale devrait permettre de séparer ces algorithmes en fonction des distributions des données.