**Laboratoire 1**

par

Alexandre Prud’Homme 7293804

Julien Desautels 7331746

Soumis à

Dr. Mok Beldjehem

dans le cadre du cours

Base de données II

CSI 3530

Université d’Ottawa

Le 11 novembre 2018

(A) Les requêtes SQL pour chaque question :

SELECT sailors.sname, sailors.age

FROM SAILORS;

SELECT DISTINCT sailors.sname, sailors.age

FROM SAILORS;

SELECT sailors.sid, sailors.sname, sailors.rating, sailors.age

FROM SAILORS

WHERE sailors.rating> 7;

SELECT sailors.sname

FROM SAILORS

INNER JOIN RESERVES ON sailors.sid = reserves.sid

WHERE reserves.bid = 103;

SELECT sailors.sname

FROM SAILORS

INNER JOIN RESERVES ON sailors.sid = reserves.sid

INNER JOIN BOATS ON reserves.bid = boats.bid

WHERE boats.color = 'Red';

SELECT DISTINCT boats.color

FROM BOATS

INNER JOIN RESERVES ON boats.bid = reserves.bid

INNER JOIN SAILORS ON reserves.sid = sailors.sid

WHERE sailors.sname = 'Lubber';

SELECT sailors.sname

FROM sailors

INNER JOIN reserves ON sailors.sid = reserves.sid;

SELECT sailors.sname, sailors.rating + 1

FROM sailors

INNER JOIN reserves r1 ON sailors.sid = r1.sid

INNER JOIN reserves r2 ON sailors.sid = r2.sid

WHERE r1.day = r2.day AND r1.bid <> r2.bid;

SELECT sailors.age

FROM sailors

WHERE sailors.sname LIKE 'B\_%b';

SELECT sailors.sname

FROM sailors

INNER JOIN reserves ON sailors.sid = reserves.sid

INNER JOIN boats ON reserves.bid = boats.bid

WHERE (boats.color = 'Red') or (boats.color = 'Green');

SELECT s1.sname

FROM sailors s1, reserves r1, boats b1

WHERE s1.sid = r1.sid AND r1.bid = b1.bid AND b1.color = 'Red'

INTERSECT

SELECT s2.sname

FROM sailors s2, reserves r2, boats b2

WHERE s2.sid = r2.sid AND r2.bid = b2.bid AND b2.color = 'Green';

SELECT s1.sid

FROM sailors s1, reserves r1, boats b1

WHERE s1.sid = r1.sid AND r1.bid = b1.bid AND b1.color = 'Red'

EXCEPT

SELECT s2.sid

FROM sailors s2, reserves r2, boats b2

WHERE s2.sid = r2.sid AND r2.bid = b2.bid AND b2.color = 'Green';

SELECT sailors.sid

FROM sailors

WHERE sailors.rating = 10

UNION

SELECT reserves.sid

FROM reserves

WHERE reserves.bid = 104;

SELECT AVG(sailors.age)

FROM sailors;

SELECT AVG(sailors.age)

FROM sailors

WHERE sailors.rating = 10;

SELECT sailors.sname, sailors.age

FROM sailors

WHERE sailors.age = (SELECT MAX(sailors.age)

FROM sailors);

SELECT COUNT(sailors.sid)

FROM sailors;

SELECT COUNT(DISTINCT sailors.sname)

FROM sailors;

SELECT MIN(sailors.age), sailors.rating

FROM sailors

GROUP BY sailors.rating;

SELECT MIN(sailors.age), sailors.rating

FROM sailors

WHERE sailors.age>= 18

GROUP BY sailors.rating

HAVING COUNT(sailors) >= 2;

(B)Les plans d’évaluation pour chaque requête SQL :









































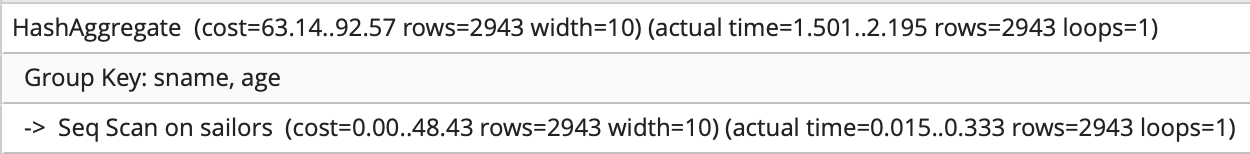
(C)

Les chemins d’accès et les méthodes d’implémentations des opérateurs impliqués dans chaque plan d’évaluation pour chaque requête SQL :

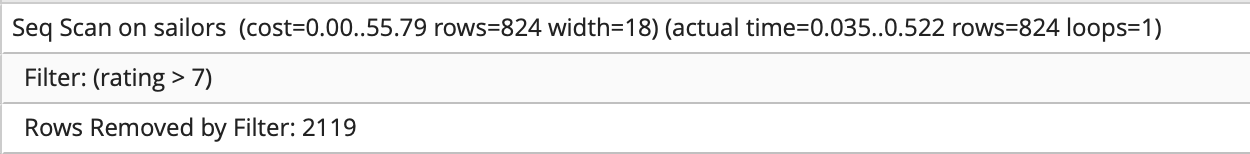
1. Un scan séquentiel est utilisé sur la table « sailors » et un retour de « sname » et « age ». « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 0.775 ms.



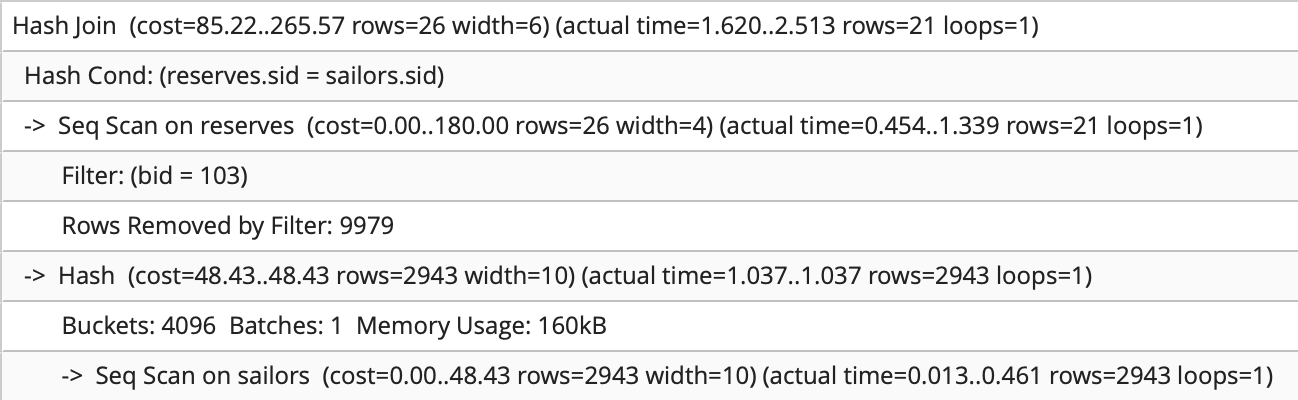
1. Un agrégat de hachage est utilisé afin d’éliminer les tuples identiques. L’agrégat de hachage a besoins d’un opérateur d’agrégat et une clé en groupe. Dans ce cas, la clé en groupe est « sname » et « age ». Un scan séquentiel est utilisé et le retour de l’information est fait.« EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 2.546 ms.



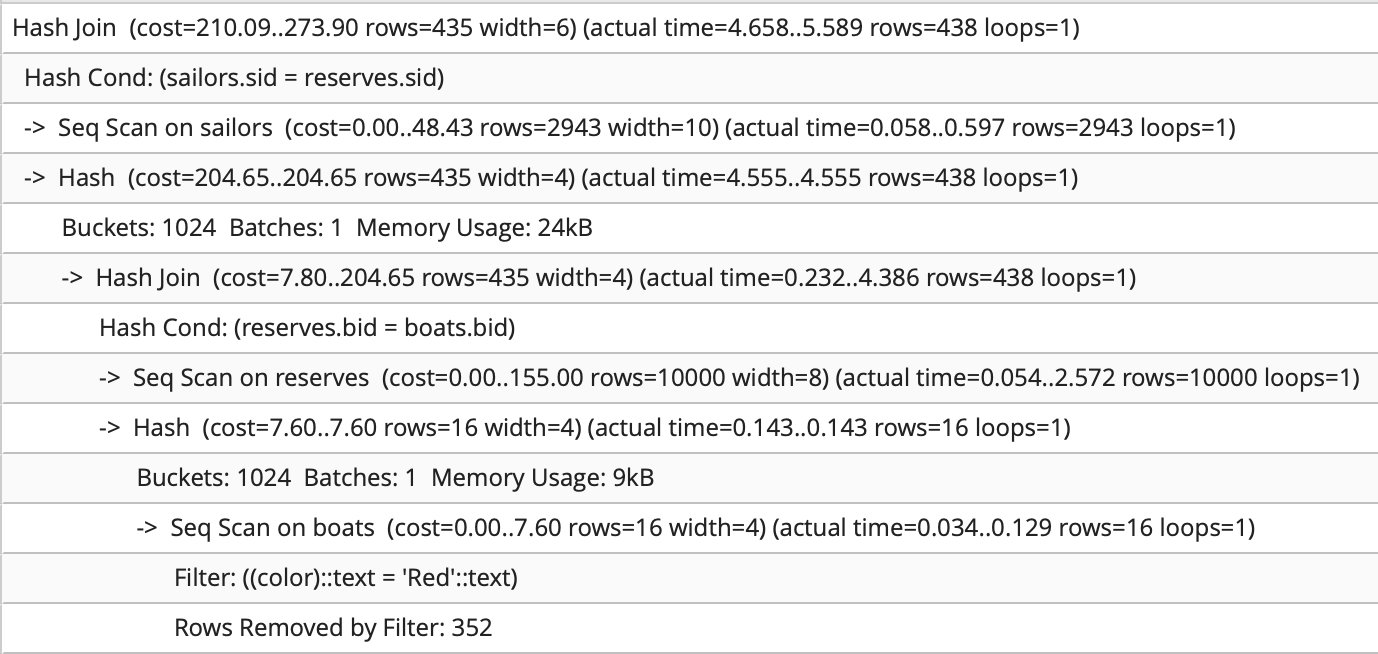
1. Un scan séquentiel avec un filtre « rating > 7 » est utilisé pour retourner l’information. « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 0.613 ms.



1. Avec la condition de hachage « sailors.sid = reserves.sid » créer une jointure de hachage. Par la suite, faire un scan séquentiel sur la table « reserves » avec le filtre « reserves.bid = 103 ». Avec le hash faire un scan séquentiel de « sailors » pour retourner l’information. « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 2.592 ms.



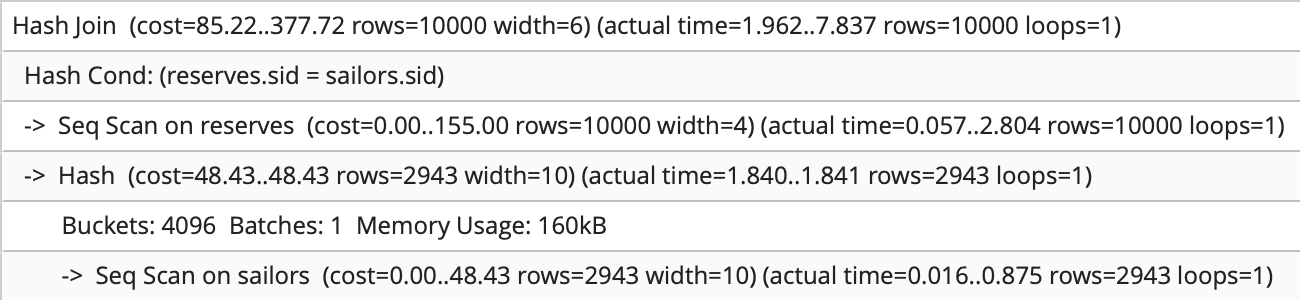
1. Créer une jointure de hachage avec la condition de hachage « sailors.sid = reserves.sid ». Par la suite, faire un scan séquentiel de la table « sailors ». Créer une deuxième jointure de hachage avec la condition de hachage « reserves.bid = boats.bid ». Par la suite, faire un scan séquentiel de la table « réserves ». Finalement, faire un scan séquentiel de la table « boats » afin de retourner les données. « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 5.727 ms.



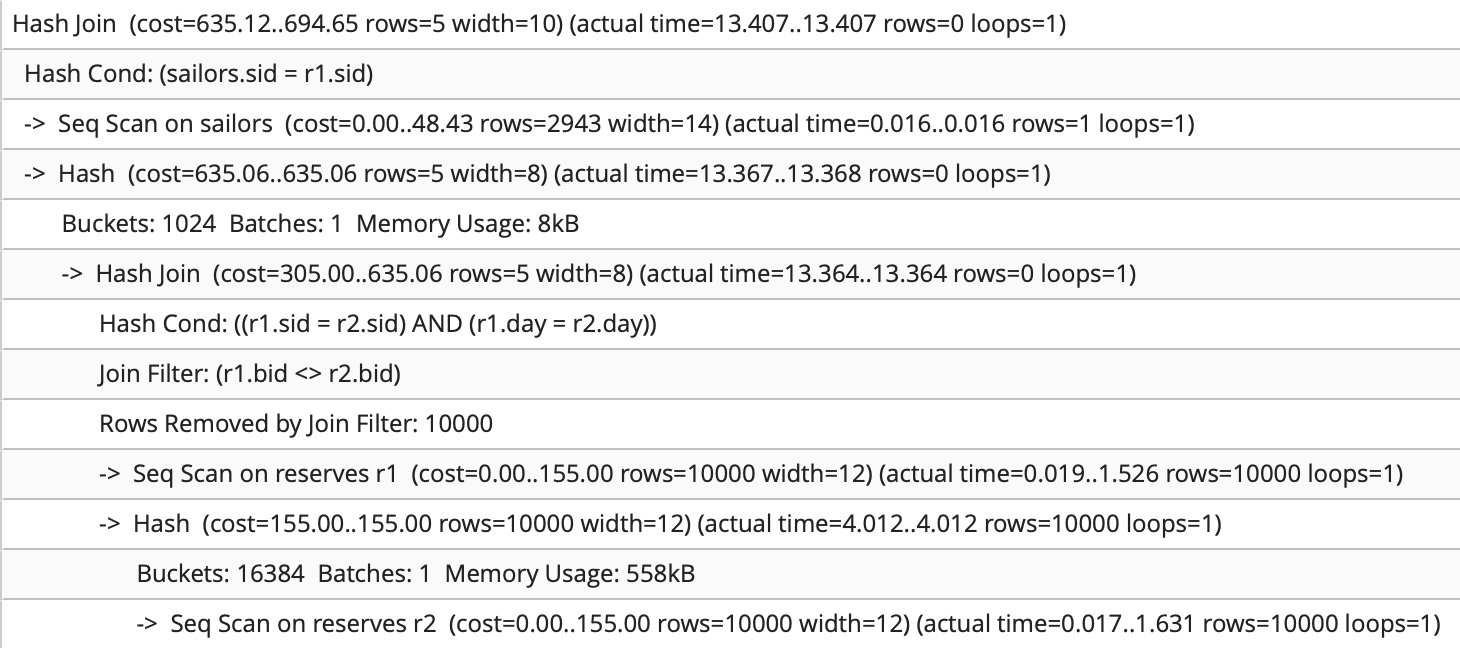
1. On commence avec un quicksortde la table « boats » avec la clé de triage « boats.color ». Par la suite, il y a une jointure hash de la condition de hachage « boats.bid = reserves.bid » et un scan séquentiel de « boats ». Par la suite, une deuxième jointure hash avec la condition de hachage « reserves.sid = sailors.sid » et un scan séquentiel de la table « reserves ». Par la suite, un scan séquentiel de la table « sailors » avec le filtre « sailors.sname = ‘Lubber’ » afin de retourner les données. « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 0.924 ms.



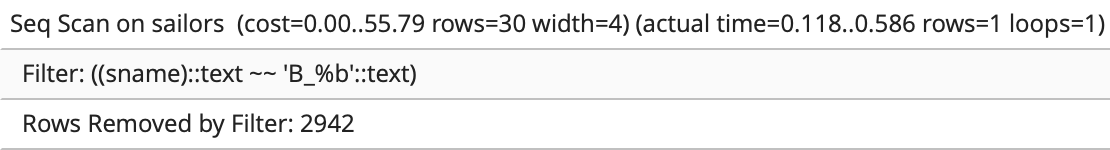
1. Créer une jointure de hachage avec la condition de hachage « sailors.sid = reserves.sid ». Par la suite, faire un scan séquentiel de la table « reserves ». Finalement, faire un scan séquentiel de la table « sailors » afin de retourner les valeurs. « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 8.599 ms.



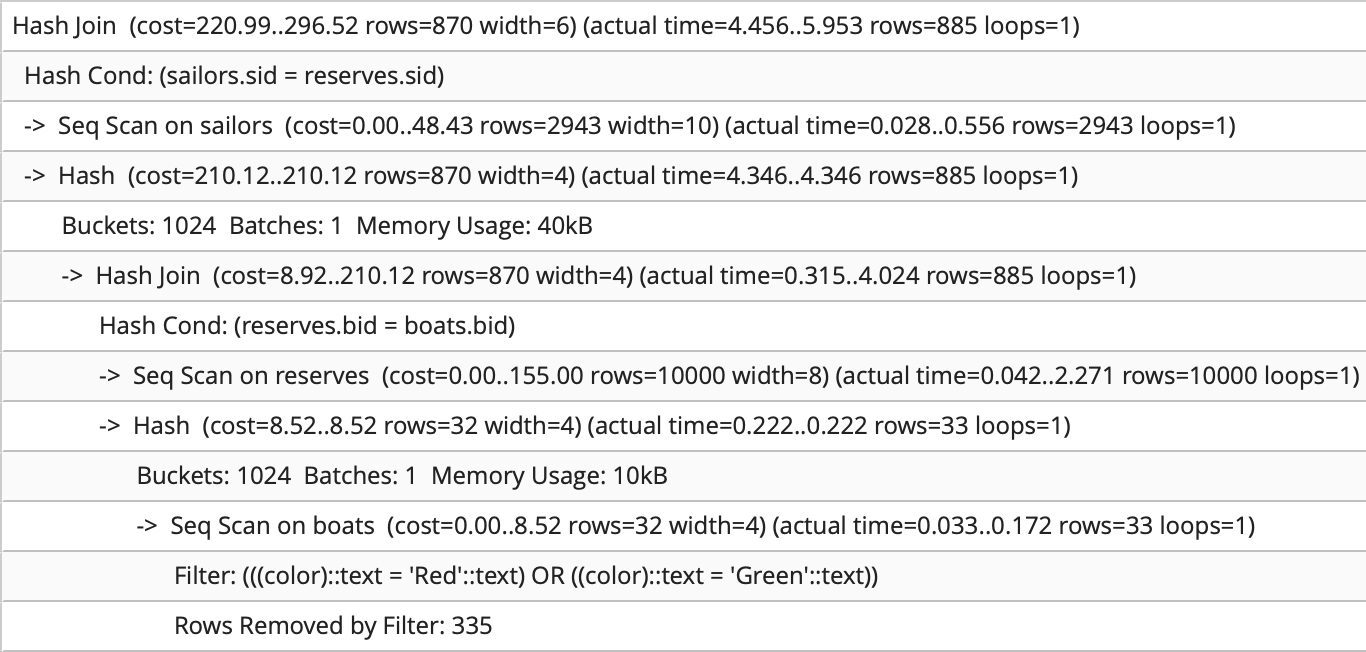
1. Créer une jointure de hachage avec la condition de hachage « sailors.sid = r1.sid ». Par la suite, faire un scan séquentiel de la table « sailors ». Créer une deuxième jointure de hachage avec la condition « (r1.day = r2.day) AND (r1.bid <> r2.bid) » et le filtre de jointure « r1.bid <> r2.bid ». Ensuite, faire un scan séquentiel de la table « reserves r1 » et finalement un dernier scan séquentiel de la table « reserves r2 » afin d’obtenir le résultat. « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 13.688 ms.



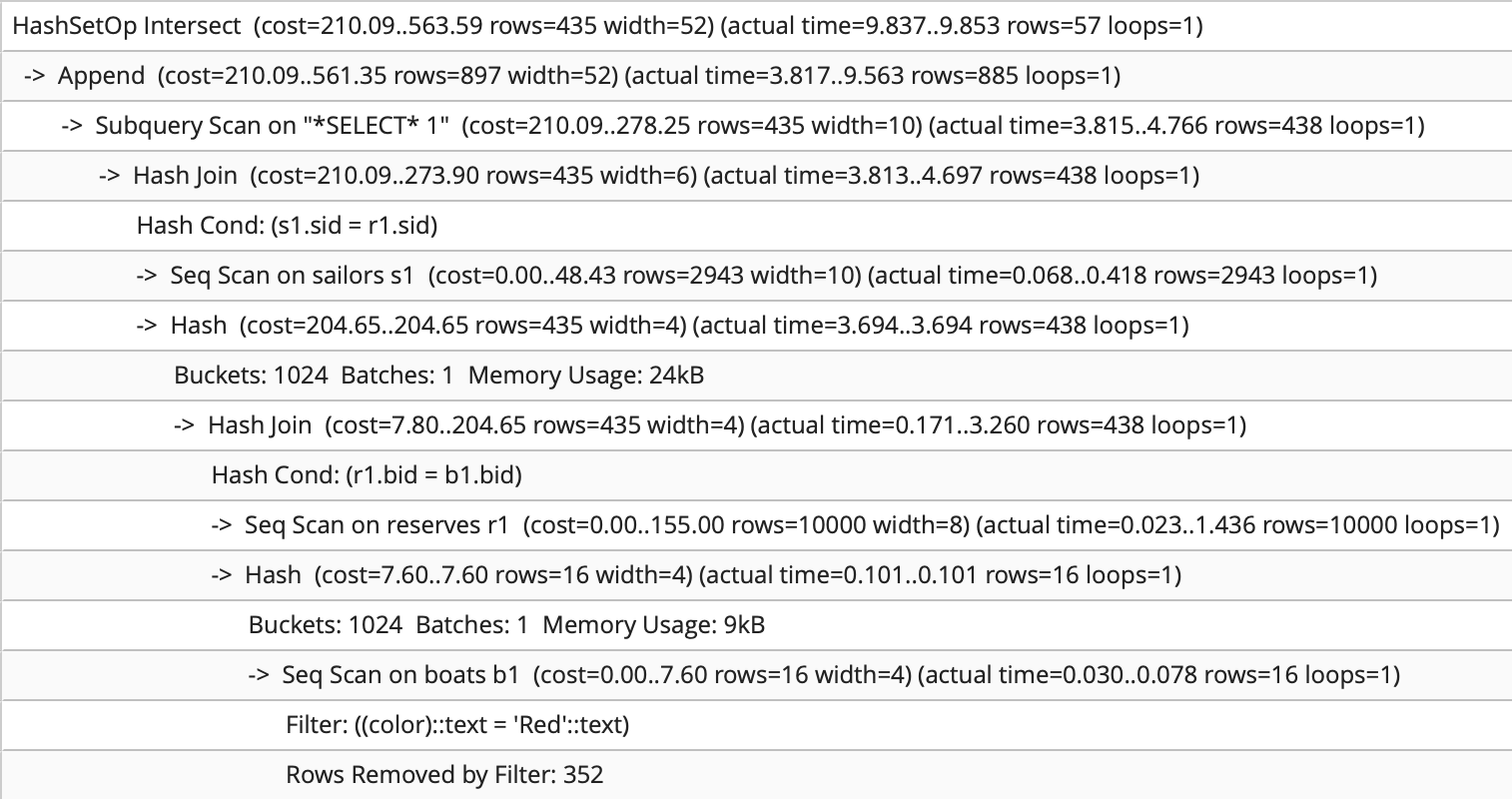
1. Utiliser un scan séquentiel de la table « sailors » avec le filtre « sailors.sname LIKE ‘B\_%b’ ». « ‘B\_%b’ » indique un mot qui commence avec le caractère « B », un caractère variable « \_ », n’importe quel nombre de caractères « % » et qui termine avec « b ». Alors, la longueur du mot est au minimum 3 lettres. « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 0.685 ms.

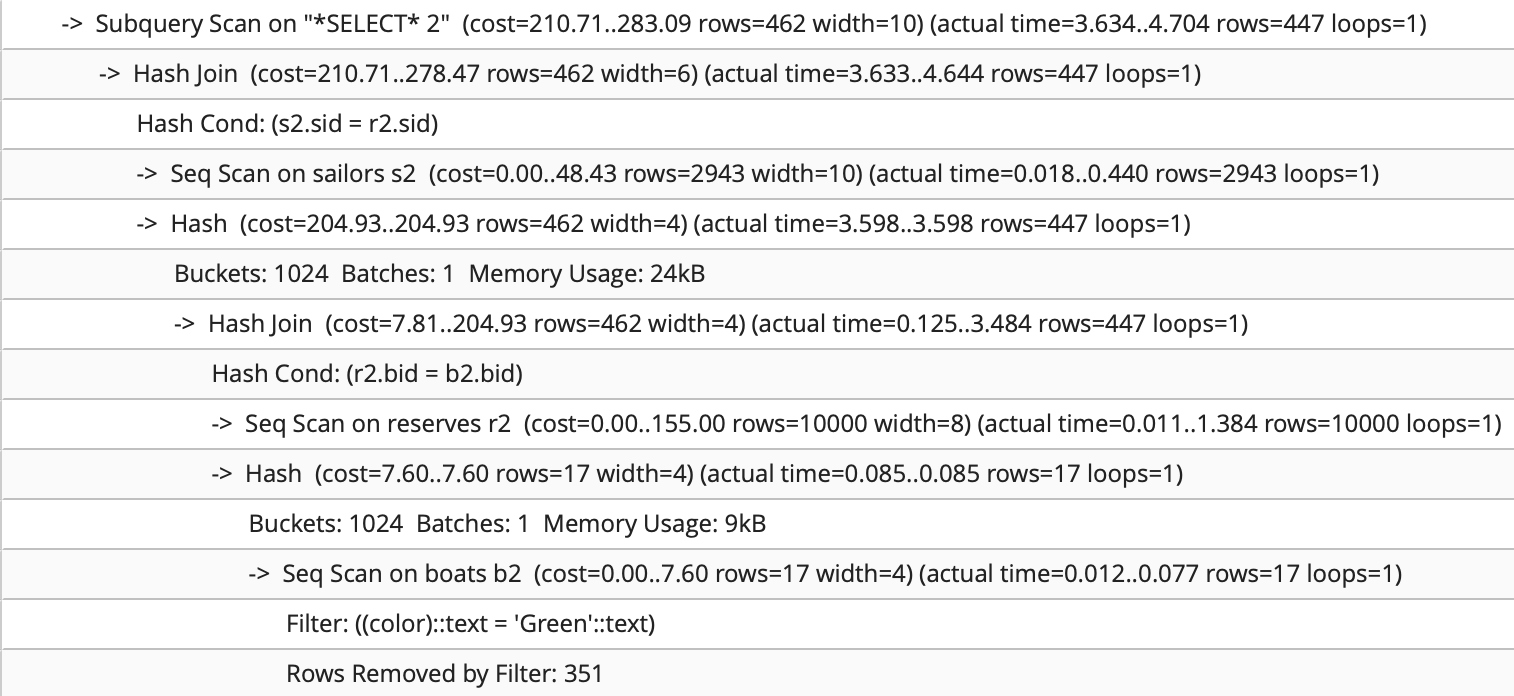


1. Créer une jointure de hachage avec la condition de hachage « sailors.sid = reserves.sid ». Par la suite, faire un scan séquentiel de la table « sailors ». Créer une deuxième jointure de hachage avec la condition de hachage « reserves.bid = boats.bid ». Ensuite, faire un scan séquentiel de la table « reserves » et finalement un dernier scan séquentiel de la table « boats » avec le filtre « (boats.color = ‘Red’) OR (boats.color = ‘Green’) ». « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 6.198 ms.

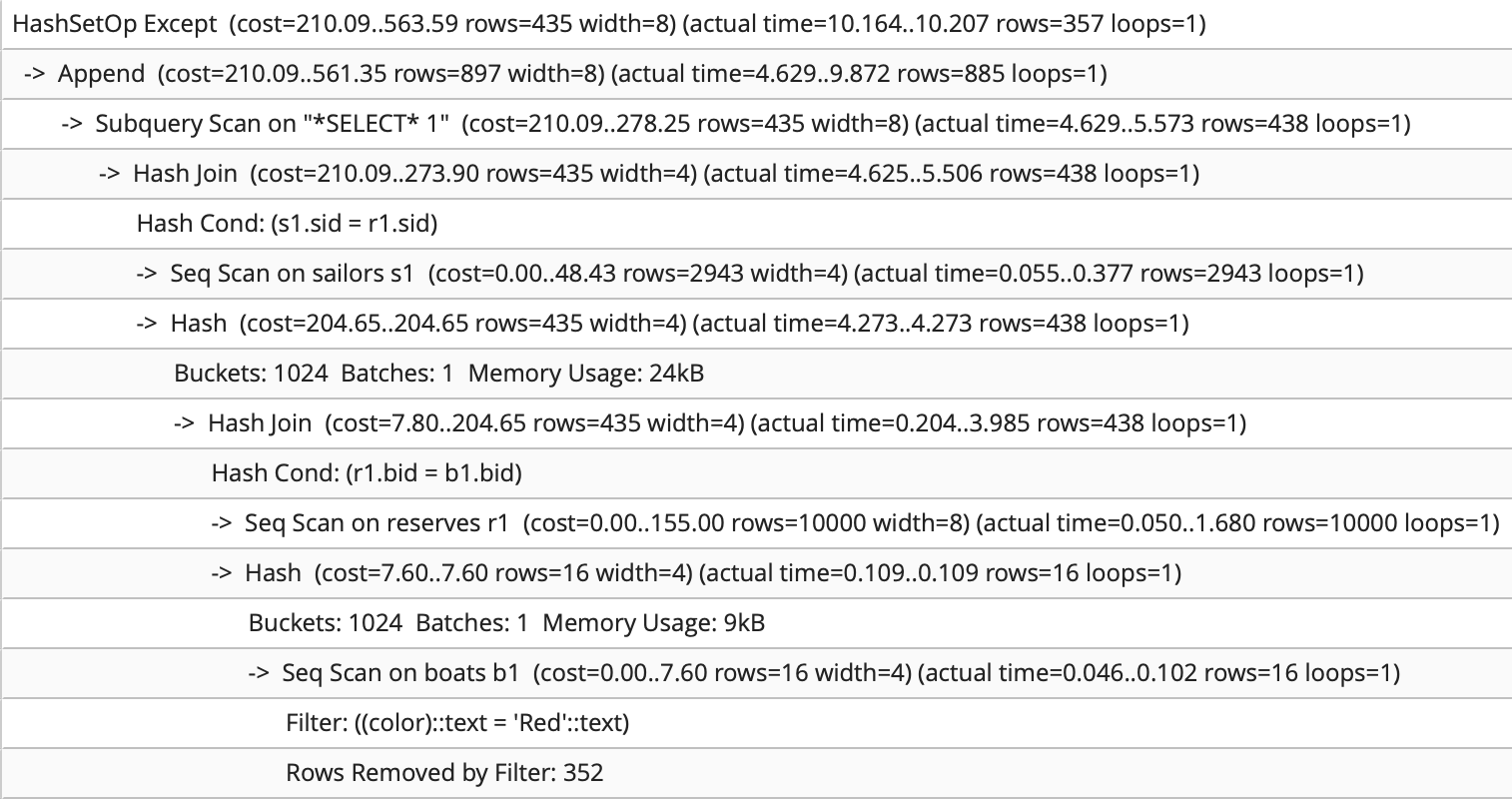


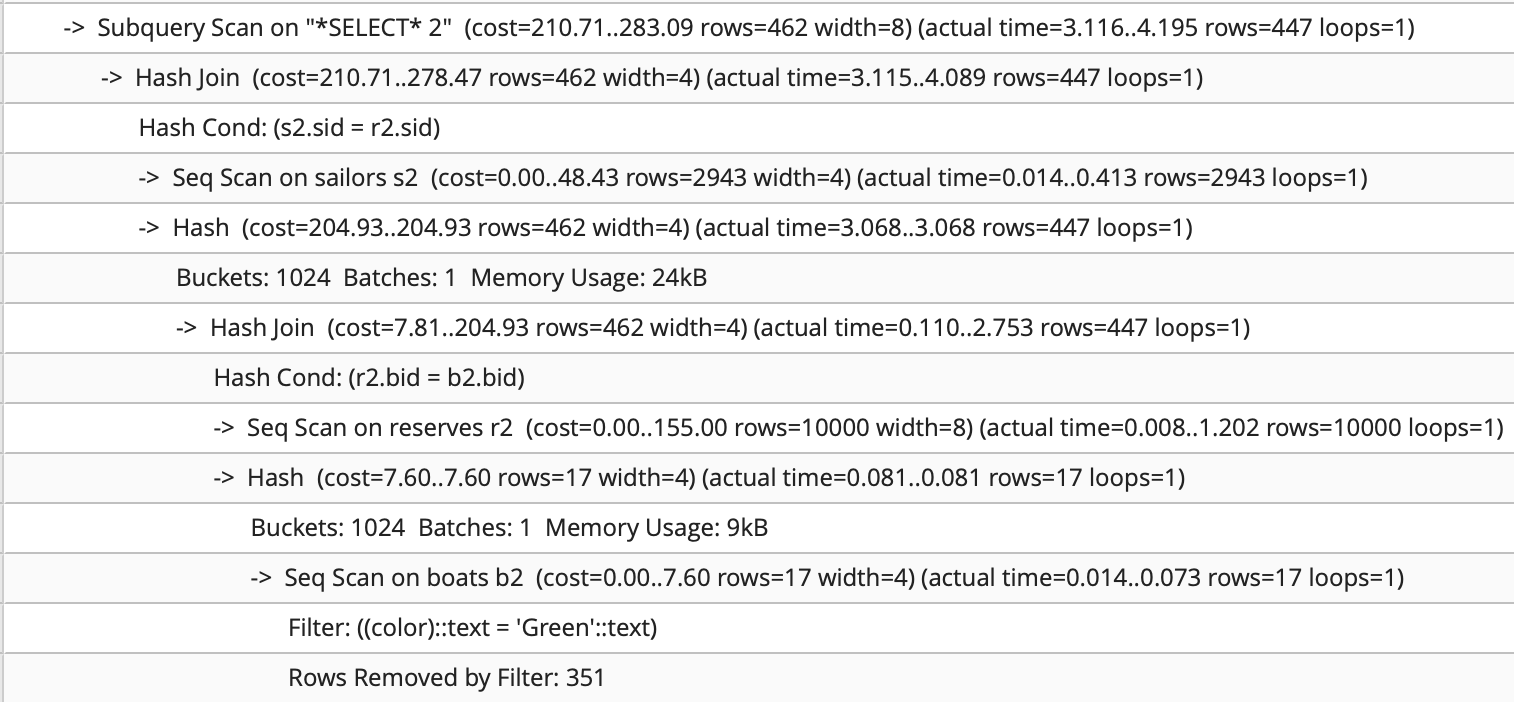
1. On commence avec l’intersection de deux requêtes. L’opération de l’intersection est hachée. Avec la première requête, il faut faire in scan de sous-requête. Par la suite, une jointure de hachage avec la condition « s1.sid = r1.sid ». Ensuite un scan séquentiel de la table s1 de « sailors ».Un hachage suivit d’une deuxième jointure de hachage avec la condition de hachage « r1.bid = b1.bid » est requise. Un scan séquentiel de la table « reserves » et un hachage est requis. Finalement, un scan séquentiel de la table « boats » avec le filtre dont la couleur doit être rouge. La deuxième requête suit exactement les mêmes étapes que la première requête. Par contre, à la fin, le filtre cherche une sélection avec la couleur verte. « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 10.105 ms.



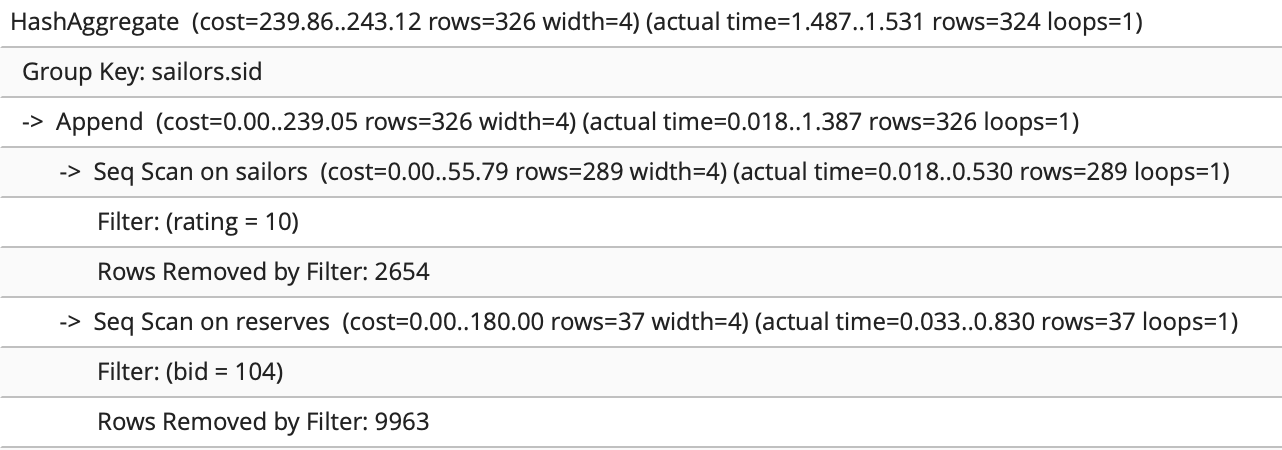


1. On commence avec l’exception de deux requêtes. L’opération de l’intersection est hachée. Avec la première requête, il faut faire in scan de sous-requête. Par la suite, une jointure de hachage avec la condition « s1.sid = r1.sid ». Ensuite un scan séquentiel de la table s1 de « sailors ». Un hachage suivit d’une deuxième jointure de hachage avec la condition de hachage « r1.bid = b1.bid » est requise. Un scan séquentiel de la table « reserves » et un hachage est requis. Finalement, un scan séquentiel de la table « boats » avec le filtre dont la couleur doit être rouge. La deuxième requête suit exactement les mêmes étapes que la première requête. Par contre, à la fin, le filtre cherche une sélection avec la couleur verte. « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 10.757 ms.

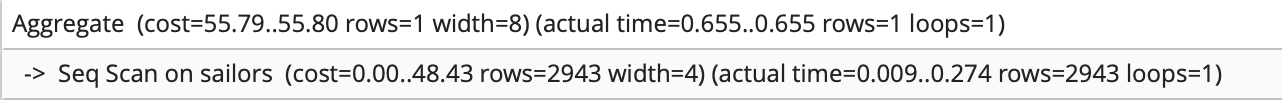




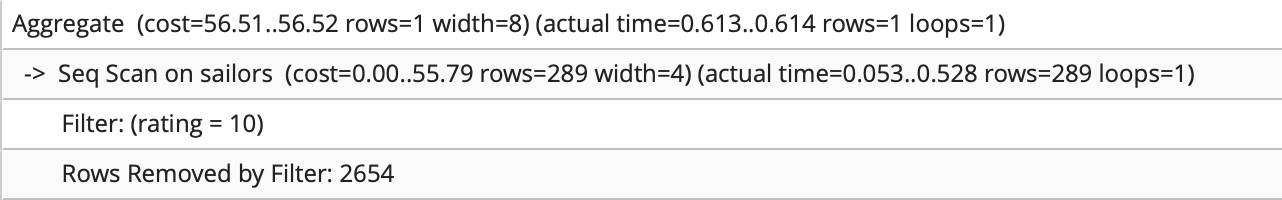
1. Un agrégat de hachage avec une clé en groupe « sailors.sid » est utilisé. Par la suite, il y a l’union entre le scan séquentiel de « sailors » avec le filtre « sailors.rating = 10 » et le scan séquentiel de « reserves » avec le filtre « reserves.bid = 104 ». « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 1.575 ms.



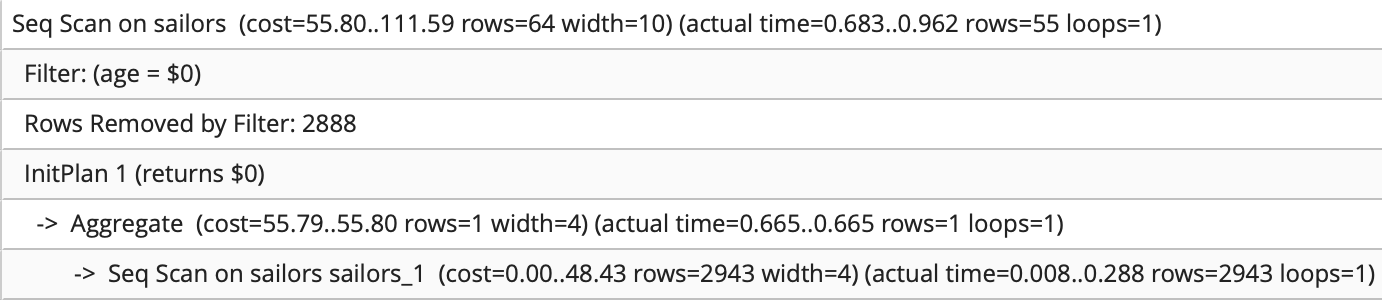
1. L’agrégat de la moyenne est utilisé. Alors, un scan séquentiel de « sailors » est utilisé pour retourner la moyenne de l’âge. « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 0.696 ms.



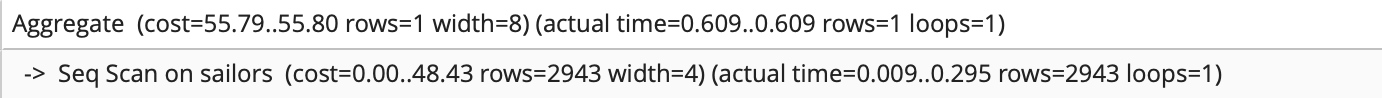
1. L’agrégat de la moyenne est utilisé. Alors, un scan séquentiel de « sailors » est utilisé avec un filtre de « sailors.rating = 10 » pour retourner la moyenne de l’age. « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 0.716 ms.



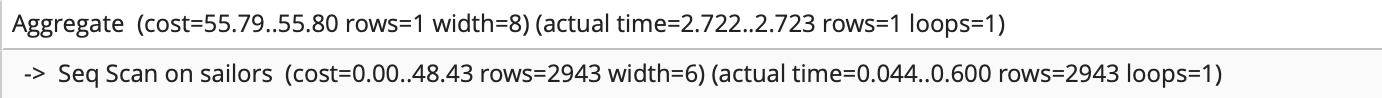
1. La requête est de forme imbriquée. Alors, il y a un scan séquentiel de la table « sailors » avec filtre. Le filtre est la requête imbriquée. Alors, cette requête imbriquée utilise l’agrégat du maximum de l’âge selon le scan séquentiel de la table « sailors ». « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 1.002 ms.



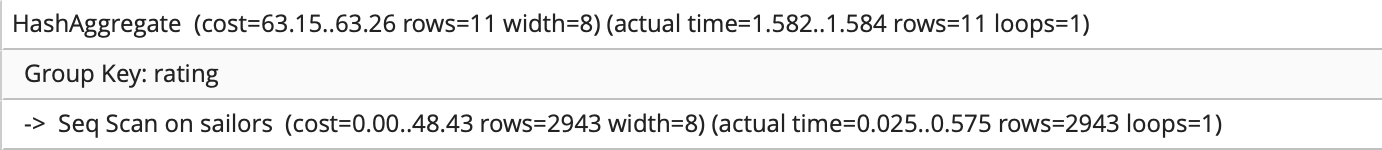
1. L’agrégat de la somme est utilisé de « sailors.sid ». Alors, un scan séquentiel de la table « sailors » est utilisé pour calculer la somme. « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 0.640 ms.



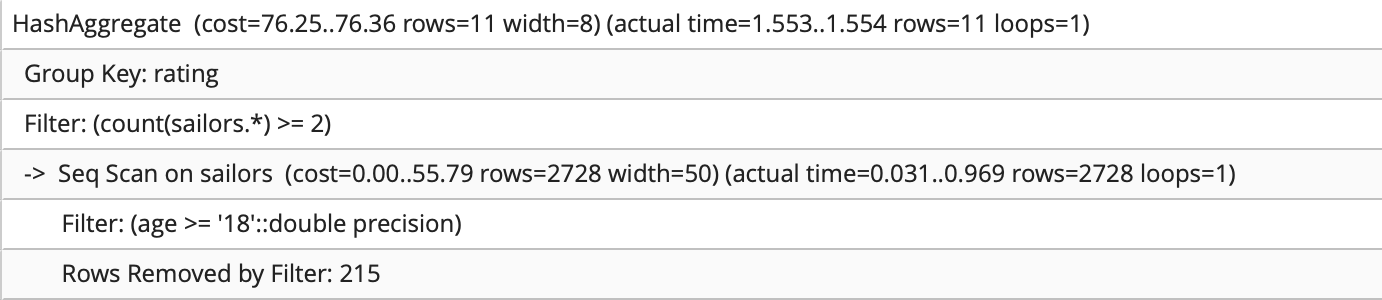
1. L’agrégat de la somme est utilisé de « sailors.sname ». Selon « distinct » les tuples en doubles sont supprimés lors du scan séquentiel de la table « sailors ». « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 2.800 ms.



1. L’agrégat de hachage est utilisé avec la clé en groupe « sailors.rating ». L’agrégat du minimum est utilisé pour « sailors.age ». Alors, il y a un scan séquentiel de la table « sailors » afin de retourner l’information. « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 1.645 ms.



1. L’agrégat de hachage est utilisé avec la clé en groupe « sailors.rating » et le filtre de la somme des « sailors » qui est supérieur ou égal à deux. Par la suite, un scan séquentiel avec le filtre « age>= 18 » est appliqué à la table « sailors ». « EXPLAIN ANALYZE » démontre ceci avec un temps d’exécution de 1.586 ms.



(D)

1. Aucune stratégie d’indexage est nécessaire pour cette requête puisqu’il faut faire une itération complète de la table « sailors ». En conséquence, le temps d’exécution des deux plans sont semblables.

|  |  |
| --- | --- |
| Premier essai sans index(ms) | Deuxième essai sans index (ms) |
| 0.775 | 0.672 |

1. La stratégie d’indexage utilisé estun arbre B+ clusteré où l’index est le « sname » de « sailors ».

CREATE INDEX SnameOfSailorsIndex ON sailors USING btree (sname);

CLUSTER sailors USING SnameOfSailorsIndex;

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 2.546 | 2.294 |

1. La stratégie d’indexage utilisé est un arbre B+ clusteré où l’index est le « rating » de « sailors ».

CREATE INDEX RatingOfSailorsIndex ON sailors USING btree (rating);

CLUSTER sailors USING RatingOfSailorsIndex;

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 0.613 | 0.416 |

1. La stratégie d’indexage utilisé est un index hash de « bid » de la table « réserves ».

CREATE INDEX BidOfReservesIndex ON reserves USING hash (bid);

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 2.592 | 1.178 |

1. La stratégie utilisée est un arbre B+ clusteré où l’index est la couleur des bateaux. Par la suite, un index de hachage du « bid » de « reserves » et un index de hachage du « sid » de la table « sailors ».

CREATE INDEX ColorOfBoatsIndex ON boats USING btree (color);

CLUSTER boats USING ColorOfBoatsIndex;

CREATE INDEX BidOfReservesIndex ON reserves USING hash (bid);

CREATE INDEX SidOfSailorsIndex ON sailors USING hash (sid);

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 5.727 | 3.619 |

1. Trois stratégies d’indexage sont utilisées. Un index hash non clusteré de « sname » de la table « sailors ». Un index hashnon clusteré de « sid » de la table « reserves ». Un index hash non clusteré de « bid » de la table « boats ».

CREATE INDEX NameOfSailorsIndex ON sailors USING hash (sname);

CREATE INDEX SidOfReservesIndex ON reserves USING hash (sid);

CREATE INDEXBidOfBoatsIndex ON boats USING hash (bid);

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 0.924 | 0.140 |

1. La stratégie d’indexage utilisé est un index de hachage non clusteré de « sid » de la table « sailors ».

CREATE INDEX SidOfSailorsIndex ON sailors USING hash (sid);

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 9.403 | 7.481 |

1. La stratégie d’indexage utilisé est un index de hachage non clusteré de « sid » de la table « reserves ».

CREATE INDEX SidOfReservesIndex ON reserves USING hash (sid);

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 13.688 | 9.162 |

1. La stratégie d’indexage utilisé est un arbre B+ clusteré où l’index est « sname » de la table « sailors ».

CREATE INDEX SnameOfSailorsIndex ON sailors USING btree (sname);

CLUSTER sailors USING SnameOfSailorsIndex;

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 0.685 | 0.100 |

1. Trois stratégies d’indexage sont utilisées. Un index hash non clusteré de « color » de la table « boats ». Un index hash non clusteré de « bid » de la table « reserves ». Un index hash non clusteré de « sid » de la table « reserves ».

CREATE INDEX ColorOfBoatsIndex ON boats USING hash (color);

CREATE INDEX BidOfReservesIndex ON reserves USING hash (bid);

CREATE INDEX SidOfReservesIndex ON reserves USING hash (sid);

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 6.198 | 3.766 |

1. Trois stratégies d’indexage sont utilisées. Un index hash non clusteré de « color » de la table « boats ». Un index hash non clusteré de « bid » de la table « reserves ». Un index hash non clusteré de « sid » de la table « reserves ».

CREATE INDEX ColorOfBoatsIndex ON boats USING hash (color);

CREATE INDEX BidOfReservesIndex ON reserves USING hash (bid);

CREATE INDEX SidOfReservesIndex ON reserves USING hash (sid);

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 10.105 | 6.858 |

1. Trois stratégies d’indexage sont utilisées. Un index hash non clusteré de « color » de la table « boats ». Un index hash non clusteré de « bid » de la table « reserves ». Un index hash non clusteré de « sid » de la table « reserves ».

CREATE INDEX ColorOfBoatsIndex ON boats USING hash (color);

CREATE INDEX BidOfReservesIndex ON reserves USING hash (bid);

CREATE INDEX SidOfReservesIndex ON reserves USING hash (sid);

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 10.105 | 8.243 |

1. Deux stratégies d’indexage sont utilisées. Un index hash non clusteré de « rating » de la table « sailors ». Un index hash non clusteré de « bid » de la table « reserves ».

CREATE INDEX RatingOfSailorsIndex ON sailors USING hash (rating);

CREATE INDEX bidOfReservesIndex ON reserves USING hash (bid);

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 1.575 | 0.695 |

1. Aucune stratégie d’indexage est nécessaire pour cette requête puisqu’il faut faire une itération complète de la table « sailors ». En conséquence, le temps d’exécution des deux plans sont semblables.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 0.696 | 0.693 |

1. La stratégie d’indexage d’un index hash non clusteré de « rating » de la table « sailors ».

CREATE INDEX RatingOfSailorsIndex ON sailors USING hash (rating);

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 0.716 | 0.242 |

1. La stratégie d’indexage est un arbre B+ clusteré où l’index est l’âge de la table « sailors ».

CREATE INDEX AgeOfSailorsIndex ON sailors USING btree (age);

CLUSTER sailors USING AgeOfSailorsIndex;

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 1.002 | 0.893 |

1. La stratégie d’indexage est un arbre B+ clusteré où l’index est « sid » de la table « sailors ».

CREATE INDEX SidOfSailorsIndexBtree ON sailors USING btree (sid);

CLUSTER sailors USING SidOfSailorsIndexBtree;

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 0.640 | 0.606 |

1. La stratégie d’indexage est un arbre B+ clusteré où l’index est « sid » de la table « sailors ».

CREATE INDEX SidOfSailorsIndex ON sailors USING btree (sid);

CLUSTER sailors USING SidOfSailorsIndex;

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 2.800 | 1.653 |

1. La stratégie d’indexage est un arbre B+ clusteré où l’index est « rating » de la table « sailors ».

CREATE INDEX RatingOfSailorsIndex ON sailors USING btree (rating);

CLUSTER sailors USING RatingOfSailorsIndex;

Avec cette stratégie, le temps d’exécution est plus rapide.

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 1.645 | 1.389 |

1. La stratégie d’indexage est un arbre B+ clusteré où l’index est « age » de la table « sailors ».

CREATE INDEX AgeOfSailorsIndex ON sailors USING btree (age);

CLUSTER sailors USING AgeOfSailorsIndex;

|  |  |
| --- | --- |
| Sans index (ms) | Index utilisé (ms) |
| 1.586 | 1.541 |

Avec cette stratégie, le temps d’exécution est plus rapide.

Il faut noter que pour la section (D) chaque optimisation a été conçue spécifiquement pour la requête en question. La raison pourquoi on ne peut ne pas combiner toute les stratégies d’indexations d’un coup est que s’il y a plusieurs clusters c’est seulement le dernier cluster qui est pris en compte. Alors, exécuter toutes les stratégies en même temps pourrait ne pas optimiser le temps de réponse de la requête. Il faut noter que toutes les stratégies sont présentes dans le fichier SQL mais, ils devraient seulement être exécutés de la même façon qu’elles sont affichées dans le rapport.