Advanced DataBases C4 Group Bank Management System

Samuel Lúcio Vicente, 251720 — Daniel Silva, 251702 — Jorge Marrero Camiruaga, 251438

Politechnika Wrocławska today

Improvements

Query 1

\mathbf{SQL}

```
UPDATE Account
    SET amount = amount*2
 3
    WHERE amount <= ALL(
 4
           SELECT Avg(amount)
 5
           FROM Account
 6
    );
 7
    SELECT Name
 8
 9
       FROM Costumer
10
        INNER JOIN Person
11
           ON GovID = PersonGovID
12
        INNER JOIN Account
13
           ON CostumerID = CostumerCostumerID
14
        WHERE amount >= ALL(
15
           SELECT MAX(amount)
16
           FROM Account
17
    );
```

First Improvement

• B-Tree index on Account based on the amount

Times

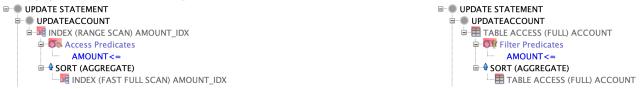
	Max	Min	Avg	#
Before	3,49	2,88	3,086	5
After	4,98	4,56	4,788	5

Observations

Longer times with the INDEX enabled.

This might seem counterintuitive has now in the UPDATE statement we see that instead of a TABLE ACCESS FULL we see a INDEX RANGE SCAN, which should be faster since less rows are being fetched, although thats true when using a INDEX in UPDATE statements and in our case updating the column on which the INDEX

is based on brings with it more memory accesses and CPU time to keep the INDEX upto date with the changes being made. If we check the stats for this execution with and without the INDEX we see that there is a lot more physical read total IO requests for when the INDEX is in use, 46 vs 225 when not in use.



The SELECT statement gets better with the index, as it should since there is now a INDEX RANGE SCAN instead of a TABLE ACCESS FULL when joining the Account table. We see that in the stats the physical read total IO requests for when the INDEX is in use is 8 vs 14 when not in use, we also see that in the session logical reads we have much more accesses to the memory per second 892 vs 216 when the index is in use.



Although we see that the SELECT statement got better with the INDEX its improvements aren't enough to cover the performance hit the UPDATE query got.

Second Improvement

• Partition Account by range of amount

Times

	Max	Min	Avg	#
Before	3,49	2,88	3,086	5
After	18,6	13,73	15,202	5

Observations

Again longer times with partition.

We see that there is Partition pruning but since we are updating the column that the partition is based on then it has to move the new values to their new partitions which takes time and negates the effort of partitioning the tables.

■ UPDATE STATEMENT		
UPDATE	ACCOUNT	
PARTITION RANGE		ITERATOR
☐ TABLE ACCESS	ACCOUNT	FULL
□ ○ ▼ Filter Predicates		
AMOUNT<=		
□ • SORT		AGGREGATE
PARTITION RANGE		ALL
TABLE ACCESS	ACCOUNT	FULL

The SELECT statement gets better with the partition, as it should since there is now a PARTITION RANGE ITERATOR instead of a TABLE ACCESS FULL when joining the Account table. We see that in the stats the physical read total IO requests for when the PARTITION is in use is 0 vs 14 when not in use, we also see that in the session logical reads we have much more accesses to the memory per second 892 vs 617 when the PARTITION is in use.

M HASH JOIN			
□ O™ Access Predicates			
COSTUMER.COSTUMERID=COS	TUMERCOSTUMERID		
■ Mested loops			
STATISTICS COLLECTOR			
PARTITION RANGE		ITERATOR	KEY
☐ TABLE ACCESS	ACCOUNT	FULL	KEY
□ O♥ Filter Predicates			
ACCOUNT.AMOUNT	「>=		
■ PARTITION RANGE		ALL MIN/MAX	
⊟ • • SORT		AGGREGATE	
TABLE ACCESS	ACCOUNT	FULL	

From this we can conclude that using an INDEX or a PARTITION based on the column we are updating is a bad practice since it forces the INDEX/PARTITION to rearrange to keep track of changes which is very expensive. From this we can gather also that keeping PARTITIONs upto date is much more expensive than keeping INDEXs upto date.

Query 2

\mathbf{SQL}

```
ALTER SYSTEM FLUSH BUFFER_CACHE;
2
    ALTER SYSTEM FLUSH SHARED_POOL;
3
    SET TIMING ON;
4
    select count(*)
5
6
    from savingsaccount inner join Account
7
       on AccountAccountID = AccountID
8
    inner join Costumer
9
       on CostumerCostumerID = CostumerID
10
    inner join Person
11
       on PersonGovID = GovID
12
    where durationyears>7 and Nationality = 'Poland';
13
    update SavingsAccount
14
15
    set interestRate =
    CASE
16
       WHEN durationYears>7 THEN interestrate + 0.03
17
       ELSE interestrate + 0.01
18
19
    END;
```

Third Improvement

- List partition in Person based on the nationalities
- Range partition in SavingsAccount based on duration years, one partition for 7 and one for ¿7

Times

	Max	Min	Avg	#
Before	lol	lol	lol	5
After	lol	lol	lol	5

Observations

lol

lol

lol

Query 3

\mathbf{SQL}

```
DECLARE
1
2
        costID NUMBER:=1;
3
    BEGIN
4
5
6
    WHILE costID<700
7
     UPDATE Account
9
    SET amount =
10
     CASE
11
         WHEN (
         SELECT EXTRACT(YEAR FROM CURRENT_DATE) - EXTRACT(YEAR FROM (
12
13
             SELECT BeginDate
             FROM Account INNER JOIN SavingsAccount
14
               ON AccountID=AccountAccountID
15
16
             WHERE AccountAccountID=(
17
               select min(AccountID)
18
               from Account inner join SavingsAccount
19
                   on AccountAccountID=AccountID
20
               where CostumerCostumerID=costID)))
21
           AS year FROM dual) > (
22
             SELECT DurationYears
23
             FROM SavingsAccount
24
             WHERE AccountAccountID=(
25
               select min(AccountID)
26
               from Account inner join SavingsAccount
27
                   on AccountAccountID=AccountID
28
               where CostumerCostumerID=costID))
29
         THEN amount + (
30
             SELECT (amount+1)*12*DurationYears*InterestRate
31
             FROM SavingsAccount INNER JOIN Account
32
             ON AccountID=AccountAccountID
33
             WHERE AccountAccountID=(
34
               select min(AccountID)
35
               from Account inner join SavingsAccount
36
                   on AccountAccountID=AccountID
37
               where CostumerCostumerID=costID))
38
         ELSE amount + (
39
         SELECT amount
40
         FROM SavingsAccount INNER JOIN Account
41
         ON AccountID=AccountAccountID
42
         WHERE AccountAccountID=(
43
               select min(AccountID)
               from Account inner join SavingsAccount
44
45
                   on AccountAccountID=AccountID
46
               where CostumerCostumerID=costID))
47
     F.ND
48
    WHERE AccountID = (
49
       SELECT AccountID
50
        FROM CurrentAccount INNER JOIN Account
           ON AccountID=AccountAccountID
51
52
        WHERE AccountAccountID=(
53
               select min(AccountID)
               from Account inner join CurrentAccount
54
                   on AccountAccountID=AccountID
55
               where CostumerCostumerID=costID));
56
57
```

```
INSERT INTO Transfer(TransferDate, Amount, AccountAccountIDFrom, AccountAccountIDTo) VALUES (
         59
     Select
      CASE
60
          WHEN (
61
62
              SELECT EXTRACT(YEAR FROM CURRENT_DATE) - EXTRACT(YEAR FROM (
63
                  SELECT BeginDate
                  FROM Account INNER JOIN SavingsAccount
64
65
                    ON AccountID=AccountAccountID WHERE AccountAccountID=(
66
                select min(AccountID)
                from Account inner join SavingsAccount
67
                    on AccountAccountID=AccountID
68
69
                where CostumerCostumerID=costID)))
70
              AS year FROM dual) > (
71
                  SELECT DurationYears
72
                  FROM SavingsAccount
73
                  WHERE AccountAccountID=(
74
                select min(AccountID)
75
                from Account inner join SavingsAccount
76
                    on AccountAccountID=AccountID
 77
                where CostumerCostumerID=costID))
 78
          THEN amount*12*DurationYears*InterestRate
79
          ELSE amount
80
      F.ND
     FROM SavingsAccount INNER JOIN Account
81
82
            ON Account ID=Account Account ID
83
         WHERE AccountAccountID=(
                select min(AccountID)
84
85
                from Account inner join SavingsAccount
86
                    on AccountAccountID=AccountID
                where CostumerCostumerID=costID)),(
87
88
                select min(AccountID)
89
                from Account inner join SavingsAccount
90
                    on AccountAccountID=AccountID
91
                where CostumerCostumerID=costID), (
92
                select min(AccountID)
93
                from Account inner join CurrentAccount
94
                    on AccountAccountID=AccountID
95
                where CostumerCostumerID=costID));
96
97
     UPDATE Account
98
     SET amount = 0
99
     WHERE AccountID = (
100
         SELECT AccountID
101
        FROM SavingsAccount INNER JOIN Account
102
            ON AccountID=AccountAccountID
103
         WHERE AccountAccountID=(
104
                select min(AccountID)
105
                from Account inner join SavingsAccount
                    on AccountAccountID=AccountID
106
107
                where CostumerCostumerID=costID));
108
     costID:=costID+1;
     END LOOP;
109
     END;
110
```

Fourth Improvement

• Allocate SavingAccount and CurrentAccount in same disk and Account in another disk

Times

	Max	Min	Avg	#
Before	lol	lol	lol	5
After	lol	lol	lol	5

Observations

lol

lol

lol

Query 4

\mathbf{SQL}

```
DECLARE
 1
 2
       empID NUMBER := 1;
 3
    BEGIN
 4
 5
    WHILE empID<5000
 6
        INSERT INTO RoleHistory(RoleRoleID, EmployeeEmployeeID, BranchBranchId, BeginDate, EndDate) VALUES
 7
 8
        1,
 9
        empID,
        (SELECT BranchBranchID FROM RoleHistory WHERE EmployeeEmployeeID = empID AND EndDate is NULL),
10
        CURRENT_DATE,
11
       NULL);
12
13
       UPDATE RoleHistory
14
           SET EndDate = CURRENT_DATE
15
        WHERE HistoryID = (SELECT min(HistoryID) FROM RoleHistory WHERE EmployeeEmployeeID = empID AND
16
            \hookrightarrow EndDate is NULL);
17
18
    empID:=empID+1;
19
    END LOOP;
20
    END;
```

Fifth Improvement

• B-Tree index on Account based on the amount

Times

	Max	Min	\mathbf{Avg}	#
Before	lol	lol	lol	5
After	lol	lol	lol	5

Observations

lol

lol

lol

Instict = Predator Avoidance + Prey Chasing

$$\begin{split} &\operatorname{PredatorAvoidance} = \sum (\overline{(-(predator.x-animal.x), -(predator.y-animal.y)} \times (animal.fov - || (-(predator.x-animal.x), (predator.y-animal.y) \times (state || (-(predator.x-animal.y)) \times (state$$

Sixth Improvement

• BitMap index in RoleHistory based in EndDate of RoleHistory if null or not

Times

	Max	Min	Avg	#
Before	lol	lol	lol	5
After	lol	lol	lol	5

Observations

lol

lol

lol

Seventh Improvement

- List partition in RoleHistory based on the EndDate being null or not
- B-Tree index based on the EmployeeEmployeeID

Times

	Max	Min	Avg	#
Before	lol	lol	lol	5
After	lol	lol	lol	5

Observations

lol

lol

lol