Advanced DataBases C4 Group
Bank Management System

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

Politechnika Wrocławska
today

Improvements

Query 1

SQL

```
UPDATE Account
SET amount = amount*2
WHERE amount <= ALL(
       SELECT Avg(amount)
       FROM Account
);

SELECT Name
    FROM Costumer
    INNER JOIN Person
        ON GovID = PersonGovID
    INNER JOIN Account
        ON CostumerID = CostumerCostumerID
    WHERE amount >= ALL(
        SELECT MAX(amount)
        FROM Account
);
```

First Improvement

- B-Tree index on Account based on the amount

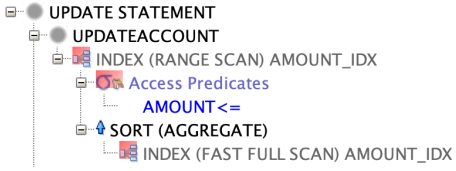
Times

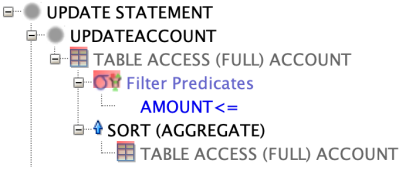
	Max	Min	Avg	#
Before	3,49	2,88	3,086	5
After	4,98	4,56	4,788	5

Observations

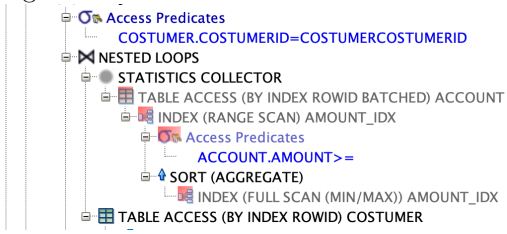
Longer times with the INDEX enabled.
This might seem counterintuitive has now in the UPDATE statement we see that instead of a TABLE ACCESS FULL we see a INDEX RANGE SCAN, which should be faster since less rows are being fetched, although thats true when using a INDEX in UPDATE statements and in our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

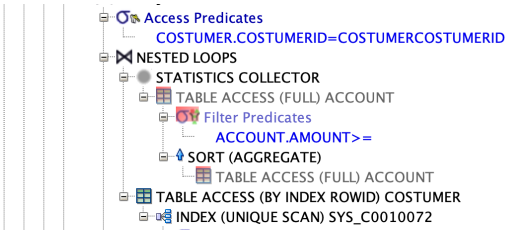
is based on brings with it more memory accesses and CPU time to keep the INDEX upto date with the changes being made. If we check the stats for this execution with and without the INDEX we see that there is a lot more physical read total IO requests for when the INDEX is in use, 46 vs 225 when not in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

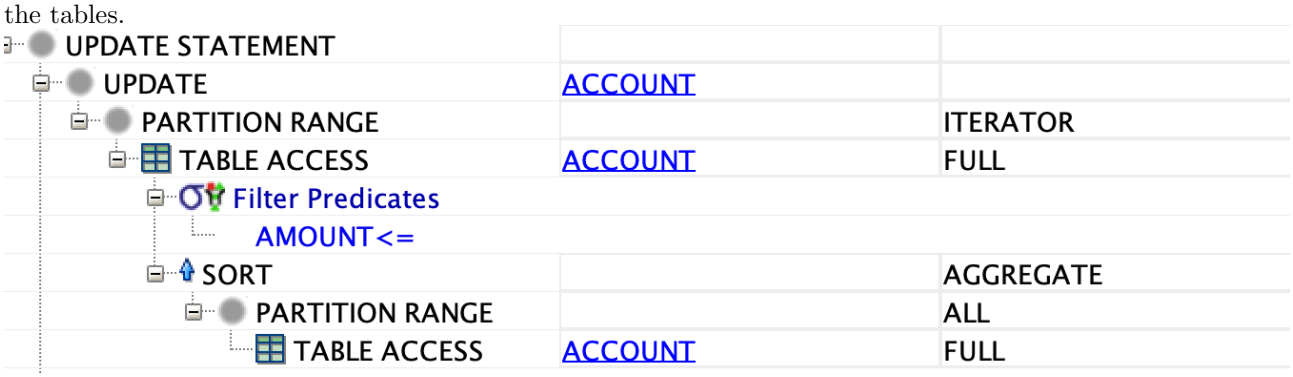
- Partition Account by range of amount

Times

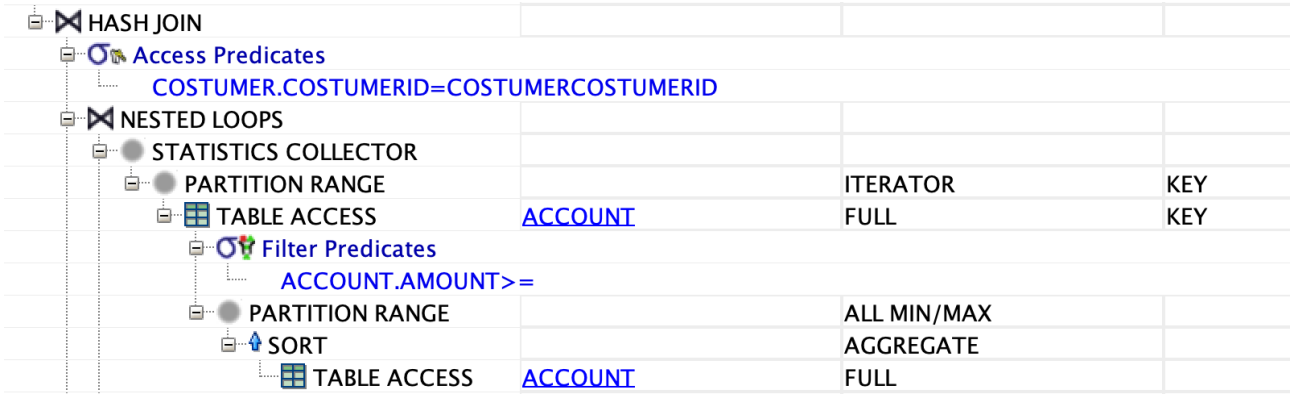
	Max	Min	Avg	#
Before	3,49	2,88	3,086	5
After	18,6	13,73	15,202	5

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.



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.



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

SQL

```
1  ALTER SYSTEM FLUSH BUFFER_CACHE;
2  ALTER SYSTEM FLUSH SHARED_POOL;
3
4  SET TIMING ON;
5  select count(*)
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
14 update SavingsAccount
15 set interestRate =
16 CASE
17     WHEN durationYears>7 THEN interestrate + 0.03
18     ELSE interestrate + 0.01
19 END;
```

Third Improvement

- List partition in Person based on the nationalities
- Range partition in SavingsAccount based on duration years, one partition for values bigger than 7 and one for smaller than

Times

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

Observations

lol
lol

lol

Query 3

SQL

```
DECLARE
    costID NUMBER:=1;

BEGIN

WHILE costID<700
LOOP
  UPDATE Account
SET amount =
  CASE
      WHEN (
      SELECT EXTRACT(YEAR FROM CURRENT_DATE) - EXTRACT(YEAR FROM (
          SELECT BeginDate
          FROM Account INNER JOIN SavingsAccount
            ON AccountID=AccountAccountID
          WHERE AccountAccountID=(
            select min(AccountID)
            from Account inner join SavingsAccount
                on AccountAccountID=AccountID
            where CostumerCostumerID=costID)))
        AS year FROM dual) > (
          SELECT DurationYears
          FROM SavingsAccount
          WHERE AccountAccountID=(
            select min(AccountID)
            from Account inner join SavingsAccount
                on AccountAccountID=AccountID
            where CostumerCostumerID=costID))
      THEN amount + (
          SELECT (amount+1)*12*DurationYears*InterestRate
          FROM SavingsAccount INNER JOIN Account
          ON AccountID=AccountAccountID
          WHERE AccountAccountID=(
            select min(AccountID)
            from Account inner join SavingsAccount
                on AccountAccountID=AccountID
            where CostumerCostumerID=costID))
      ELSE amount + (
      SELECT amount
      FROM SavingsAccount INNER JOIN Account
      ON AccountID=AccountAccountID
      WHERE AccountAccountID=(
            select min(AccountID)
            from Account inner join SavingsAccount
                on AccountAccountID=AccountID
            where CostumerCostumerID=costID))
  END
WHERE AccountID = (
    SELECT AccountID
    FROM CurrentAccount INNER JOIN Account
        ON AccountID=AccountAccountID
    WHERE AccountAccountID=(
            select min(AccountID)
            from Account inner join CurrentAccount
                on AccountAccountID=AccountID
            where CostumerCostumerID=costID));
```

```
57
58 INSERT INTO Transfer(TransferDate, Amount, AccountAccountIDFrom, AccountAccountIDTo) VALUES (
       ↪ CURRENT_DATE, (
59 Select
60   CASE
61       WHEN (
62           SELECT EXTRACT(YEAR FROM CURRENT_DATE) - EXTRACT(YEAR FROM (
63               SELECT BeginDate
64               FROM Account INNER JOIN SavingsAccount
65                 ON AccountID=AccountAccountID WHERE AccountAccountID=(
66             select min(AccountID)
67             from Account inner join SavingsAccount
68                 on AccountAccountID=AccountID
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   END
81 FROM SavingsAccount INNER JOIN Account
82        ON AccountID=AccountAccountID
83     WHERE AccountAccountID=(
84             select min(AccountID)
85             from Account inner join SavingsAccount
86                 on AccountAccountID=AccountID
87             where CostumerCostumerID=costID)),(
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
106                 on AccountAccountID=AccountID
107             where CostumerCostumerID=costID));
108 costID:=costID+1;
109 END LOOP;
110 END;
```

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

SQL

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

Fifth Improvement

- Hash partition RoleHistory by EmployeeEmployeeID and separate them in diferent disks

Times

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

Observations

lol
lol
lol

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