Practicumopdracht Query-optimalisatie

In deze opdracht onderzoek je een aantal SQL-query's welke veelgemaakte fouten bevatten. Het is in veel gevallen de bedoeling dat je de performanceproblemen verklaart en/of oplossingen zoekt om de performance te verbeteren.

Gebruik het meegeleverde create script om onderstaande opdrachten te kunnen uitvoeren.

Deel 1 – Parsing time

a) Voer onderstaande query 5 maal uit. Noteer de benodigde tijd in de kolom onder de query.

```
SELECT * FROM orders WHERE order_id_char = 1200;
```

Meting	Tijd (sec)
1	1.981
2	0.048
3	0.048
4	0.048
5	0.048

b) Doe hetzelfde voor de volgende query:

```
SELECT * FROM orders WHERE order_id_char = 10;
```

Meting	Tijd (sec)
1	0.056
2	0.044
3	0.046
4	0.047
5	0.044

c) Wat is je conclusie?

De database lijkt de tabel te 'cachen', waardoor latere queries sneller worden. Daarnaast bestaat er geen index voor deze tabel, zoals weergegeven in de volgende afbeelding.

Deel 2 – Query-optimalisatie

a) Maak de tabellen “products” en “costs” aan in de database met in ieder geval de kolommen die in onderstaande query genoemd worden. Vul de tabellen met willekeurige testdata (bijvoorbeeld met behulp van je testdatagenerator).
b) Voer onderstaande query uit en noteer de benodigde tijd om de query uit te voeren in de tabel hieronder.
c) Verbeter de performance van de query en noteer ook van deze query de benodigde tijd om de query uit te voeren.

Query	Tijd (sec)	Cost

`SELECT COUNT(*)`
`FROM product p`
`WHERE prod_list_price < 1.15 * (SELECT avg(unit_cost)`
`FROM cost c`
`WHERE c.prod_id = p.prod_id);`
	0.177	6

`Geoptimaliseerde query:`
 	
`SELECT count(*)`
`FROM product p`
`INNER JOIN cost ct`
`ON(p.prod_id = ct.prod_id)`
`WHERE p.prod_list_price < 1.15 * (SELECT avg(ct.unit_cost) FROM cost);`
 	
`SELECT count(*)`
`FROM product p`
`FULL OUTER JOIN cost ct`
`ON(p.prod_id = ct.prod_id)`
`WHERE p.prod_list_price < 1.15 * (SELECT avg(ct.unit_cost) FROM cost);`
	0,054
	
	0,047
		4
		
		4

d) Verklaar aan de hand van het execution plan van de twee query’s de verschillen in benodigde tijd.

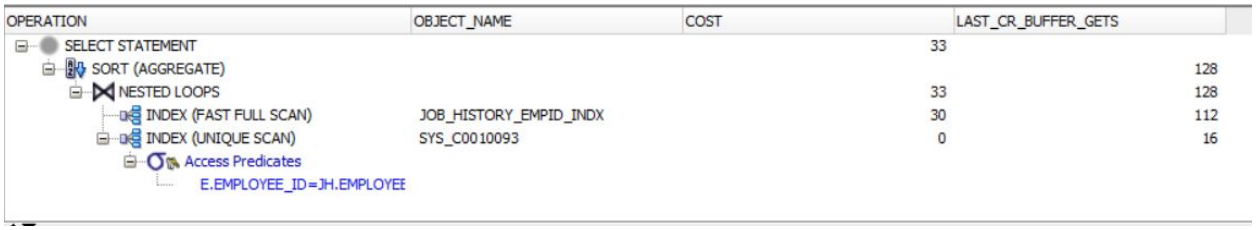
Deel 3 – Query optimalisatie

a) Voer onderstaande query uit en noteer de benodigde tijd om de query uit te voeren.
b) Verbeter de performance van de query en noteer ook van deze query de benodigde tijd om hem uit te voeren.

Query	Tijd (sec)	Cost
SELECT count(*) FROM job_history jh, employees e WHERE substr(to_char(e.employee_id),1)=substr(to_char(jh.employee_id),1);	0,103	519
Geoptimaliseerde query: SELECT count(*) FROM job_history jh, employees e WHERE e.employee_id = jh.employee_id;	0,0033	33

c) Verklaar aan de hand van het execution plan van de twee queries de verschillen in benodigde tijd.

OPERATION	OBJECT_NAME	COST	LAST_CR_BUFFER_GETS
SELECT STATEMENT		519	
SORT (AGGREGATE)			1672
HASH JOIN		519	1672
Access Predicates			
SUBSTR(TO_CHAR(E.EMPLOYEE_ID			
TABLE ACCESS (FULL)	JOB_HISTORY	426	1570
INDEX (FAST FULL SCAN)	SYS_C0010093	27	102



OPERATION	OBJECT_NAME	COST	LAST_CR_BUFFER_GETS
SELECT STATEMENT		33	
SORT (AGGREGATE)			128
NESTED LOOPS		33	128
INDEX (FAST FULL SCAN)	JOB_HISTORY_EMPID_INDX	30	112
INDEX (UNIQUE SCAN)	SYS_C0010093	0	16
Access Predicates			
E.EMPLOYEE_ID=JH.EMPLOYEE			

Er wordt gebruik gemaakt van twee indexen zodat niet de hele tabel gelezen hoeft te worden. Bij de eerdere query wordt eerst een bewerking uitgevoerd, en vervolgens de gehele tabel eventjes ingelezen en bewerkt.

Deel 4 – Query optimalisatie

a) Voer onderstaande query's uit en noteer de benodigde tijd.

Query	Tijd (sec)	Cost
select count(*) from (select name from old union select name from new);	0.116	1214
select count(*) from (select name from old union all select name from new);	0.056	887

b) Verklaar aan de hand van het execution plan van de twee queries de verschillen in tijd.

Bij de eerste (alleen union) dient er een sort (unique) gedaan te worden, omdat union (zonder all) alleen distinct waarden teruggeeft. Union all heeft dit niet.

Deel 5 – Query's combineren

a) Voer onderstaande query's uit en noteer per query de benodigde tijd om hem uit te voeren. Tel de tijden op en noteer het resultaat.

Query	Tijd (sec)	Cost

`SELECT COUNT (*)`
`FROM myemp`
`WHERE salary < 2000;`
	0.084	1785

`SELECT COUNT (*)`
`FROM myemp`
`WHERE salary BETWEEN 2000 AND 4000;`
	0.180	1785

`SELECT COUNT (*)`
`FROM myemp`
`WHERE salary>4000;`
	0.058	1785

Totaal:	0.322	5355

b) Maak een query die de drie gegeven query's combineert en dus minder vaak door dezelfde rijen hoeft te lopen. Noteer ook van deze query de benodigde tijd om hem uit te voeren. Tip: maak gebruik van het CASE WHEN statement.

Geoptimaliseerde query	Tijd (sec)	Cost

`SELECT COUNT (CASE WHEN salary < 2000`
`THEN 1 ELSE null END) count1,`
`COUNT (CASE WHEN salary BETWEEN 2001 AND 4000`
`THEN 1 ELSE null END) count2,`
`COUNT (CASE WHEN salary > 4000`
`THEN 1 ELSE null END) count3`
`FROM myemp;`
	0,117	1743

c) Vergelijk de tijd voor het uitvoeren van de losse query's opgeteld met de tijd voor de zelfgemaakte query en verklaar het verschil aan de hand van de execution plans.

In plaats van 3 keer door ALLES heen te lopen, hoeft er nu maar een keer door alles heen gelopen te worden, en kan er dan per rij een afweging worden gemaakt. Hierdoor is deze case when veel efficienter.

Deel 6 – Conversies reduceren

a) Voer onderstaande query uit en noteer de benodigde tijd om hem uit te voeren.
b) Bekijk in de database het datatype van de kolom order_id_char en vergelijk dit met de meegegeven waarde in de query.
c) Verbeter de performance van de query en noteer ook van deze query de benodigde tijd om hem uit te voeren.

Query	Tijd (sec)	Cost
`SELECT * FROM orders WHERE order_id_char = 50;`	1.612	888

`Geoptimaliseerde query:`
 	
`SELECT * FROM orders WHERE order id char = '50';`
	0.012	2

d) **Verklaar aan de hand van het execution plan van de twee query's de verschillen in tijd.**
Het verschil in tijd zit hem in het feit dat bij de eerste query telkens een TO_NUMBER uitegevoerd moet worden.
e) Heeft de INDEX op de tabel invloed op dit verschil? Zo ja, waarom?

Ja, want hij hoeft alleen maar naar de id in de index te kijken en niet door de hele tabel te lopen.

	4	7	10
Opdracht 4: Query-optimalisatie-opdracht (tweetal)	Alleen metingen uitgevoerd met matige onderbouwing.	Metingen uitgevoerd en onderbouwde antwoorden op de normale vragen.	Metingen uitgevoerd en onderbouwde antwoorden op de normale en optionele vragen.