Milestone 8 Bewijs in PDF

Jonas Van der Reysen - INF202B

Overzicht vergelijking:

Tabel info voor partitionering:

	SEGMENT_NAME	SEGMENT_TYPE	MB	TABLE_COUNT
1	LEERLINGEN	TABLE	16	400000

Query:

Deze query wordt gebruikt om te kijken (per klas) hoeveel leerlingen van elk geslacht binnen een bepaalde “batch” vallen met hun scores. Zo kun je bijvoorbeeld zien welk geslacht het meest prominent aanwezig is in bij de 90%

```
-- Stap 2: Analyse voor optimalisatie
SELECT s.naam, k.naam, l.geslacht, COUNT(l.geslacht) AS aantal_in_bereik
FROM scholen s
         JOIN klassen k ON s.schoolid = k.scholen_schoolid
         JOIN leerlingen l ON k.klasid = l.klassen_klasid
WHERE s.naam = 'School 1' AND l.score BETWEEN 90 AND 100
GROUP BY s.naam, k.naam, l.geslacht
ORDER BY k.naam;
```

Explain plan

Operation	Params	Rows	Total Cost	Raw Desc
Select		1876	560.0	cpu_cost = 200153816, io_cost = 553
Order By (SORT ORDER BY)		1876	560.0	cpu_cost = 200153816, io_cost = 553
Group By (HASH GROUP BY)		1876	560.0	cpu_cost = 200153816, io_cost = 553
Hash Join		1876	558.0	cpu_cost = 139102034, io_cost = 553
Full Scan (TABLE ACCESS FULL)	table: KLASSEN;	2000	5.0	cpu_cost = 472579, io_cost = 5
Merge Join (MERGE JOIN CARTESIAN)		37515	553.0	cpu_cost = 133977955, io_cost = 548
Full Scan (TABLE ACCESS FULL)	table: SCHOLEN;	1	3.0	cpu_cost = 40007, io_cost = 3
Sort (BUFFER SORT)		37515	550.0	cpu_cost = 133937948, io_cost = 545
Full Scan (TABLE ACCESS FULL)	table: LEERLINGEN;	37515	550.0	cpu_cost = 133937948, io_cost = 545

NA partitionering:

Partitie script + uitleg partitie sleutel

Ik heb ervoor gekozen om te partitioneren op score zodat je gemakkelijk alle leerlingen met een bepaalde score kunt ophalen uit de tabel. Bijvoorbeeld: “geef mij eens alle leerlingen die exact 50% gescoord hebben”.

```
CREATE TABLE leerlingen
(
    leerlingid     NUMBER(6) GENERATED ALWAYS AS IDENTITY (start with 1 increment by 1) NOT NULL PRIMARY KEY,
    klassen_klasid NUMBER(6)                                                            NOT NULL,
    voornaam       VARCHAR2(25),
    achternaam     VARCHAR2(50),
    geslacht       CHAR(1)
        CONSTRAINT leerlingen_geslacht_MVX CHECK ( geslacht IN ('M', 'V', 'X') ),
    klasnummer     NUMBER                                                               NOT NULL
        CONSTRAINT leerlingen_klasnummer_tss_0_40 CHECK ( klasnummer BETWEEN 1 AND 40 ),
    score          NUMBER(3, 0)
        CONSTRAINT leerlingen_score_percentage_tss_0_100 CHECK ( score BETWEEN 0 AND 100 )
)
    PARTITION BY RANGE (score)
    INTERVAL (5)
(
    PARTITION p0 VALUES LESS THAN (5)
);
```

Tabel info NA partitionering:

	SEGMENT_NAME	SEGMENT_TYPE	MB	TABLE_COUNT
1	LEERLINGEN	TABLE PARTITION	168	400000

	SEGMENT_NAME	SEGMENT_TYPE	MB
1	LEERLINGEN	TABLE PARTITION	8
2	LEERLINGEN	TABLE PARTITION	8
3	LEERLINGEN	TABLE PARTITION	8
4	LEERLINGEN	TABLE PARTITION	8
5	LEERLINGEN	TABLE PARTITION	8
6	LEERLINGEN	TABLE PARTITION	8
7	LEERLINGEN	TABLE PARTITION	8
8	LEERLINGEN	TABLE PARTITION	8
9	LEERLINGEN	TABLE PARTITION	8
10	LEERLINGEN	TABLE PARTITION	8
11	LEERLINGEN	TABLE PARTITION	8
12	LEERLINGEN	TABLE PARTITION	8
13	LEERLINGEN	TABLE PARTITION	8
14	LEERLINGEN	TABLE PARTITION	8
15	LEERLINGEN	TABLE PARTITION	8
16	LEERLINGEN	TABLE PARTITION	8
17	LEERLINGEN	TABLE PARTITION	8
18	LEERLINGEN	TABLE PARTITION	8
19	LEERLINGEN	TABLE PARTITION	8
20	LEERLINGEN	TABLE PARTITION	8
21	LEERLINGEN	TABLE PARTITION	8

Query:

```
-- Stap 2: Analyse voor optimalisatie
SELECT s.naam, k.naam, l.geslacht, COUNT(l.geslacht) AS aantal_in_bereik
FROM scholen s
         JOIN klassen k ON s.schoolid = k.scholen_schoolid
         JOIN leerlingen l ON k.klasid = l.klassen_klasid
WHERE s.naam = 'School 1' AND l.score BETWEEN 90 AND 100
GROUP BY s.naam, k.naam, l.geslacht
ORDER BY k.naam;
```

Explain plan na partitionering

Operation	Params	Rows	Total Cost	Raw Desc
Select		2198	72.0	cpu_cost = 82635720, io_cost = 69
Order By (SORT ORDER BY)		2198	72.0	cpu_cost = 82635720, io_cost = 69
Group By (HASH GROUP BY)		2198	72.0	cpu_cost = 82635720, io_cost = 69
Hash Join		2198	70.0	cpu_cost = 21223181, io_cost = 69
Full Scan (TABLE ACCESS FULL)	table: KLASSEN;	2000	5.0	cpu_cost = 472579, io_cost = 5
Merge Join (MERGE JOIN CARTESIAN)		43960	65.0	cpu_cost = 15454603, io_cost = 64
Full Scan (TABLE ACCESS FULL)	table: SCHOLEN;	1	3.0	cpu_cost = 40007, io_cost = 3
Sort (BUFFER SORT)		43960	62.0	cpu_cost = 15414595, io_cost = 61
Unknown (PARTITION RANGE ITERATOR)		43960	62.0	cpu_cost = 15414595, io_cost = 61
Full Scan (TABLE ACCESS FULL)	table: LEERLINGEN;	43960	62.0	cpu_cost = 15414595, io_cost = 61

Conclusie:

Geef hier je deskundige conclusie van je resultaten, bekijk cost , tijd, geheugengebruik.

Na het partitioneren zie je dat de tabel zelf wel wat groter is geworden, en het inserteren van de gegevens duurt langer. Nu, wanneer je dan kijkt naar het verbruik van geheugen, opslag,.. zie je dat het de moeite waard is om 1 keer nét ietsje meer tijd en resources te gebruiken, tegenover een gigantische winst in performantie. Zo is de CPU cost drastisch gedaald. Ook de IO cost (geheugen) is van in de honderdtallen gedaald naar nog zo’n 60.

Een goede optimalisatie, dus. 🙂

IOT (om te excelleren):

Eventueel: herhaling van structuur Na partitionering, maar dan voor IOT.

IOT script + uitleg structuur tabel

<Voeg hier je IOT script in>

Tabel info NA IOT en opvullen:

Explain plan na IOT

Conclusie: