# Milestone 8 Bewijs in PDF

Jonas Van der Reysen - INF202B

# Overzicht vergelijking:

Tabel info voor partitionering:

SEGME	NT_NAME #	□ SEGMENT_TYPE	□ MB ÷	
1 LEERLING	EN	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

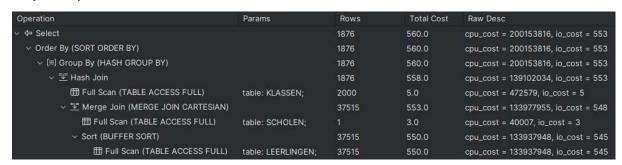
JOIN leerlingen l ON k.klasid = l.klassen_klasid

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



## 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".

### Tabel info NA partitionering:

#### 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
<ul> <li>Order By (SORT ORDER BY)</li> </ul>		2198	72.0	cpu_cost = 82635720, io_cost = 69
√ (≡) Group By (HASH GROUP BY)		2198	72.0	cpu_cost = 82635720, io_cost = 69
∨ 型 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
∨ 型 Merge Join (MERGE JOIN CARTESIAN)			65.0	cpu_cost = 15454603, io_cost = 64
☐ Full Scan (TABLE ACCESS FULL)	table: SCHOLEN;		3.0	cpu_cost = 40007, io_cost = 3
→ Sort (BUFFER SORT)		43960	62.0	cpu_cost = 15414595, io_cost = 61
<ul> <li>Unknown (PARTITION RANGE ITERATOR)</li> </ul>		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: