# Milestone 8 Bewijs in PDF

Maxim Derboven - INF203

# Overzicht vergelijking:

# Tabel info voor partitionering:

	■■ SEGMENT_NAME	■ SEGMENT_TYPE	II MB ≎	■ TABLE_COUNT ÷
1	PERFORMANCES	TABLE	19	502505

### Query:

#### Test 1

```
-- Gemiddelde dagen tussen start en einde van film performances (uit 2019) (in dagen) per hall met minder dan 4 stoelen

SELECT T.THEATHER_ID, T.NAME,H.HALL_ID, ROUND(AVG(CAST(P.ENDTIME as DATE)-CAST(P.STARTTIME AS DATE))) AS "Gemiddelde dagen per performance"

FROM THEATHERS T

JOIN HALLS H on H.THEATHER_ID = T.THEATHER_ID

JOIN PERFORMANCES P on P.HALL_ID = H.HALL_ID

WHERE H.SEAT_AMOUNT < 4 AND P.STARTTIME BETWEEN TO_DATE('01-01-2019','DD-MM-YYYY') AND TO_DATE('31-12-2019','DD-MM-YYYY')

group by T.THEATHER_ID, T.NAME, H.SEAT_AMOUNT, H.HALL_ID

ORDER BY T.THEATHER_ID DESC;
```

#### Test 2

```
SELECT T.THEATHER_ID, T.NAME, ROUND(AVG(CAST(P.ENDTIME as DATE)-CAST(P.STARTTIME AS DATE))) AS "Gemiddelde dagen per performance"

FROM THEATHERS T

JOIN HALLS H on H.THEATHER_ID = T.THEATHER_ID

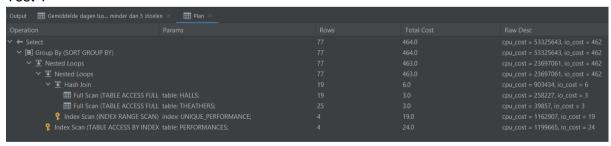
JOIN PERFORMANCES P on P.HALL_ID = H.HALL_ID

WHERE P.STARTTIME BETWEEN TO_DATE('01-11-2019 00:00','DD-MM-YYYY HH24:MI') AND TO_DATE('01-11-2019 23:59','DD-MM-YYYY HH24:MI')

GROUP BY T.THEATHER_ID, T.NAME;
```

## Explain plan

#### Test 1



#### Test 2

# NA partitionering:

Partitie script + uitleg partitie sleutel

#### Test 1

Ik partitioneer de performances elk jaar op hun start tijd ('starttime'). De eerste performance vondt plaats op 2018/01/01

```
CREATE TABLE PERFORMANCES

(

performance_id INTEGER GENERATED ALWAYS AS IDENTITY CONSTRAINT fk_performance PRIMARY KEY,

movie_id INTEGER NOT NULL CONSTRAINT fk_movie_performance REFERENCES MOVIES(MOVIE_ID) ON DELETE CASCADE,

hall_id INTEGER NOT NULL CONSTRAINT fk_hall_performance REFERENCES HALLS(HALL_ID) ON DELETE CASCADE,

starttime TIMESTAMP NOT NULL,

endtime TIMESTAMP NOT NULL

)

PARTITION BY RANGE (starttime)

INTERVAL (NUMTOYMINTERVAL(1, 'YEAR'))

(
PARTITION p0 VALUES LESS THAN (TO_DATE('2019/01/01', 'YYYY/MM/DD')),

PARTITION p1 VALUES LESS THAN (TO_DATE('2020/01/01', 'YYYY/MM/DD'))

);
```

#### Test 2

Ik partitioneer de performances elke 2 maanden op hun start tijd ('starttime'). De eerste performance vondt plaats op 2018/01/01

### Tabel info NA partitionering:

#### Test 1

	· ·						
	■■ SEGMENT_NAME		■ SEGMENT_TYPE		II≣ MB ≎	■■ TABLE_COUNT ÷	
1	PERFORMANCES		TABLE PARTITION		64	502505	
Test 2							
	■■ SEGMENT_NAME		■ SEGMENT_TYPE		II MB ≎	■■ TABLE_COUNT ÷	
1	PERFORMANCES		TABLE PARTITION		216	502505	

Query: → moet dezelfde zijn

## Explain plan na partitionering

#### Test 1

Operation	Params	Rows	Total Cost	Raw Desc
✓ ← Select				cpu_cost = 69043679, io_cost = 279
✓ [≡] Group By (SORT GROUP BY)				cpu_cost = 69043679, io_cost = 279
✓ 堻 Hash Join				cpu_cost = 39405074, io_cost = 279
✓  ▼ Merge Join				cpu_cost = 29894628, io_cost = 5
🗡 💡 Index Scan (TABLE ACCESS BY				cpu_cost = 23493, io_cost = 2
💡 Full Index Scan (INDEX FUL	L index: THEATHER_PK;			cpu_cost = 12121, io_cost = 1
➤ Sort (SORT JOIN)				cpu_cost = 29871135, io_cost = 3
Ⅲ Full Scan (TABLE ACCESS FU				cpu_cost = 258787, io_cost = 3
✓ Unknown (PARTITION RANGE SINGLE)				cpu_cost = 8501946, io_cost = 274
	) table: PERFORMANCES;	4046		cpu_cost = 8501946, io_cost = 274

### Test 2



## Conclusie:

De costs dalen omdat hij sneller doorheen de data kan door de bepaalde partitie grenzen. De tijd zal daarom ook afnemen. Hoe meer partities hoe groter de database wordt. Er wordt bij mij in eerste instantie sowieso gezocht op auto indexes (pk's) dus veel verschil is er niet.