# MileStone 8 Bewijs

# Overzicht Vergelijking:

Tabel info voor partitionering:

	■■ SEGMENT_NAME	<b>‡</b>	■ SEGMENT_TYPE ÷	÷	<b>■</b> МВ	<b>‡</b>	■■ TABLE_COUNT	<b>‡</b>
1	REVIEWS		TABLE			88		500005

#### Query:

```
SELECT /*+ NOPARALLEL(r)*/ u.EMAIL, p.NAME, TITLE, CONTENT, POSTEDDATE, LIKES

FROM REVIEWS r

JOIN USERS u ON r.REVIEW_USER_ID = u.USER_ID

JOIN SMARTPHONES p ON r.REVIEW_PHONE_ID = p.PHONE_ID

WHERE RATING = 5

GROUP BY p.NAME, u.EMAIL, TITLE, CONTENT, POSTEDDATE, LIKES

ORDER BY LIKES DESC;
```

# Explain plan

```
        Operation
        Params
        Rows
        Total Cost
        Raw Desc

        ✓ ← Select
        81744
        3174.0
        cpu_cost = 330268852, io_cost = 3164

        ✓ Unknown (PX COORDINATOR)
        81744
        3174.0
        cpu_cost = 330268852, io_cost = 3164

        ✓ Unknown (PX SEND QC (ORDER))
        81744
        3174.0
        cpu_cost = 330268852, io_cost = 3164

        ✓ Image: I
```

#### Partitie script:

De rijen worden verdeeld naargelang het aantal likes, we zijn momenteel op zoek naar de reviews met de beste rating

```
ALTER TABLE REVIEWS

MODIFY

PARTITION BY RANGE (rating) INTERVAL (1) (
PARTITION RATED_0 VALUES LESS THAN (1),
PARTITION RATED_1 VALUES LESS THAN (2),
PARTITION RATED_2 VALUES LESS THAN (3),
PARTITION RATED_3 VALUES LESS THAN (4),
PARTITION RATED_4 VALUES LESS THAN (5),
PARTITION RATED_5 VALUES LESS THAN (6));
```

# Tabel info NA partitionering

#### Query:

```
SELECT /*+ NOPARALLEL(r)*/ u.EMAIL, p.NAME, TITLE, CONTENT, POSTEDDATE, LIKES

FROM REVIEWS r

JOIN USERS u ON r.REVIEW_USER_ID = u.USER_ID

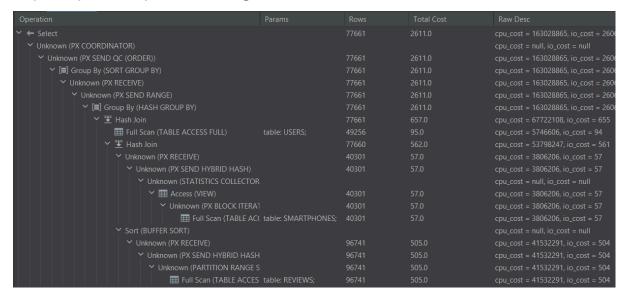
JOIN SMARTPHONES p ON r.REVIEW_PHONE_ID = p.PHONE_ID

WHERE RATING = 5

GROUP BY p.NAME, u.EMAIL, TITLE, CONTENT, POSTEDDATE, LIKES

ORDER BY LIKES DESC;
```

### Explain plan na partitionering



#### Conclusie:

Wanneer we ons baseren op de cost van beide queries zien we dat de eerste query doorheen de hele query een hoge cost heeft. Bij de tweede query zien we in het begin een hogere kost, maar deze neemt zeer snel af, waardoor de efficiëntie ook toeneemt.

De io\_cost is ook naar beneden gegaan na het partitioneren, in het eerste deel is dit ongeveer 66% van de vorige waarde, en ook dit gaat drastisch naar beneden naargelang hoe dieper we in de query gaan.

De tijd om uit te voeren is licht gedaald, maar niet zo drastisch als de cost.

Als we naar het algemene beeld kijken heeft het partitioneren er zeker voor gezorgd dat we de query efficiënter hebben kunnen uitvoeren.