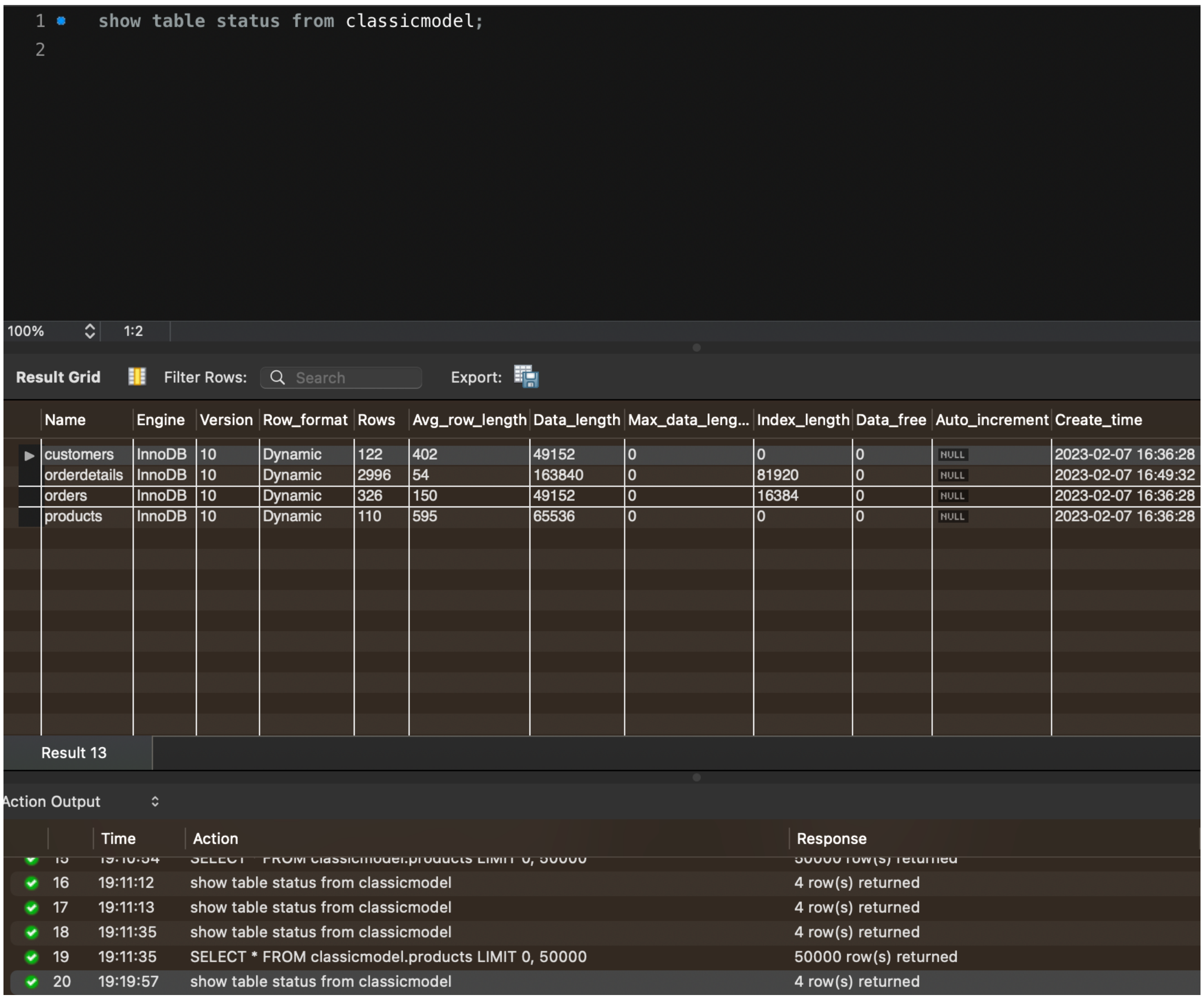
Big Data – Assignment 1

Part 2a:
The output of the command: show table status from classicmodel;



```
show table status from classicmodel;
```

100% 1:2

Result Grid | Filter Rows: Search | Export:

Name	Engine	Version	Row_format	Rows	Avg_row_length	Data_length	Max_data_leng...	Index_length	Data_free	Auto_increment	Create_time
customers	InnoDB	10	Dynamic	122	402	49152	0	0	0	NULL	2023-02-07 16:36:28
orderdetails	InnoDB	10	Dynamic	2996	54	163840	0	81920	0	NULL	2023-02-07 16:49:32
orders	InnoDB	10	Dynamic	326	150	49152	0	16384	0	NULL	2023-02-07 16:36:28
products	InnoDB	10	Dynamic	110	595	65536	0	0	0	NULL	2023-02-07 16:36:28

Result 13

Action Output

	Time	Action	Response
15	19:10:54	SELECT * FROM classicmodel.products LIMIT 0, 50000	50000 row(s) returned
16	19:11:12	show table status from classicmodel	4 row(s) returned
17	19:11:13	show table status from classicmodel	4 row(s) returned
18	19:11:35	show table status from classicmodel	4 row(s) returned
19	19:11:35	SELECT * FROM classicmodel.products LIMIT 0, 50000	50000 row(s) returned
20	19:19:57	show table status from classicmodel	4 row(s) returned

Query Analysis
Explain Analyze query (8) from. How can you improve the execution?

The output of the EXPLAIN ANALYZE command shows that the query is using a temporary table with deduplication, and performing a nested loop inner join between the orderdetails and products tables. It also shows that a covering index scan is used on the orderdetails table, and a single-row index lookup is used on the products table.

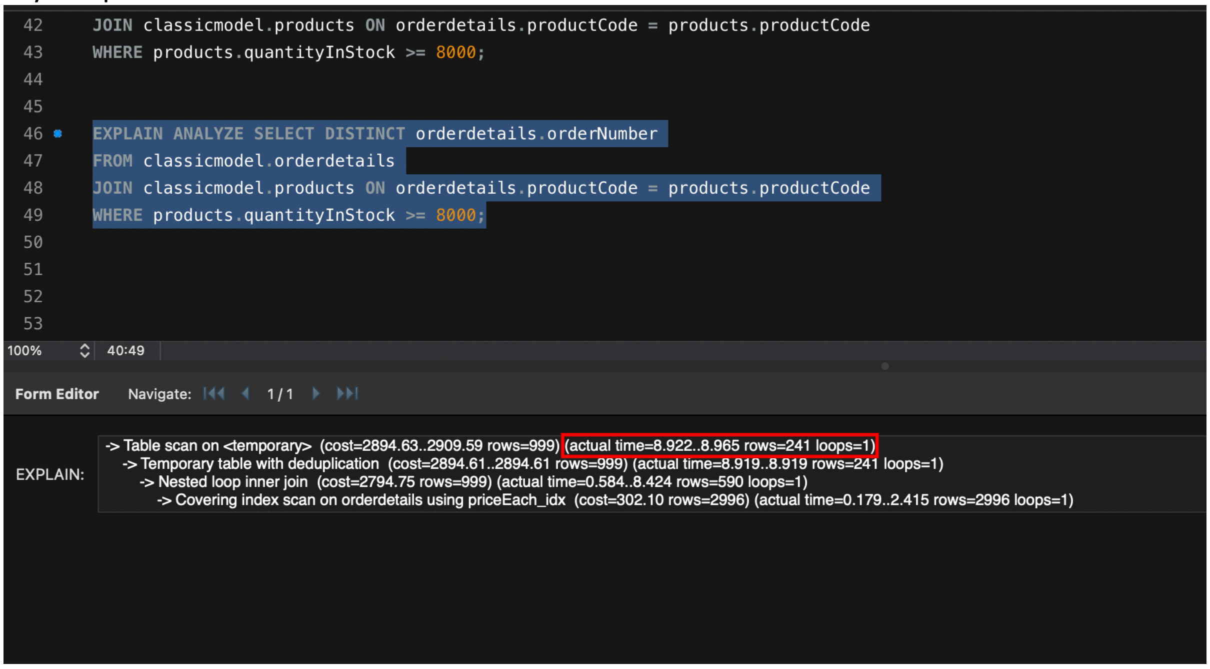
The query is currently not optimized for performance, as it is using a nested loop join which can be slow for large datasets. A possible way to improve the execution is to use a different join method, such as a hash join or a merge join, which may be faster for the given tables and data. Additionally, creating an index on the quantityInStock column of the products table may improve the performance of the query.

Commands for improving performance:

CREATE INDEX idx_orderdetails_productCodeNo ON classicmodel.orderdetails (productCode);
CREATE INDEX idx_products_quantityInStock ON classicmodel.products (quantityInStock);

ALTER TABLE classicmodel.orderdetails
ADD INDEX idx_orderdetails_products (productCode, quantityInStock);

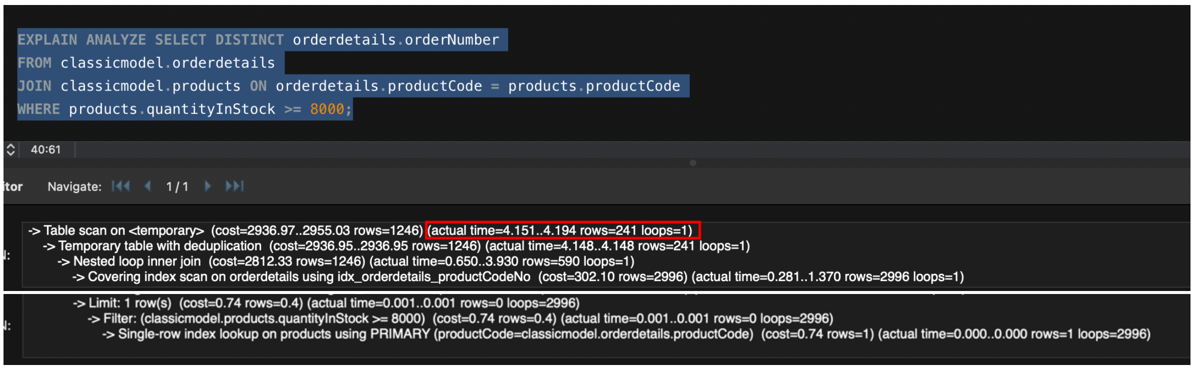
MySQL plans and Execution time before and after



```
JOIN classicmodel.products ON orderdetails.productCode = products.productCode
WHERE products.quantityInStock >= 8000;


EXPLAIN ANALYZE SELECT DISTINCT orderdetails.orderNumber
FROM classicmodel.orderdetails
JOIN classicmodel.products ON orderdetails.productCode = products.productCode
WHERE products.quantityInStock >= 8000;
```

```
-> Table scan on <temporary>  (cost=2894.63..2909.59 rows=999) (actual time=8.922..8.965 rows=241 loops=1)
    -> Temporary table with deduplication  (cost=2894.61..2894.61 rows=999) (actual time=8.919..8.919 rows=241 loops=1)
        -> Nested loop inner join  (cost=2794.75 rows=999) (actual time=0.584..8.424 rows=590 loops=1)
            -> Covering index scan on orderdetails using priceEach_idx  (cost=302.10 rows=2996) (actual time=0.179..2.415 rows=2996 loops=1)
```



```
EXPLAIN ANALYZE SELECT DISTINCT orderdetails.orderNumber
FROM classicmodel.orderdetails
JOIN classicmodel.products ON orderdetails.productCode = products.productCode
WHERE products.quantityInStock >= 8000;
```

```
-> Table scan on <temporary>  (cost=2936.97..2955.03 rows=1246) (actual time=4.151..4.194 rows=241 loops=1)
    -> Temporary table with deduplication  (cost=2936.95..2936.95 rows=1246) (actual time=4.148..4.148 rows=241 loops=1)
        -> Nested loop inner join  (cost=2812.33 rows=1246) (actual time=0.650..3.930 rows=590 loops=1)
            -> Covering index scan on orderdetails using idx_orderdetails_productCodeNo  (cost=302.10 rows=2996) (actual time=0.281..1.370 rows=2996 loops=1)
            -> Limit: 1 row(s)  (cost=0.74 rows=0.4) (actual time=0.001..0.001 rows=0 loops=2996)
                -> Filter: (classicmodel.products.quantityInStock >= 8000)  (cost=0.74 rows=0.4) (actual time=0.001..0.001 rows=0 loops=2996)
                    -> Single-row index lookup on products using PRIMARY (productCode=classicmodel.orderdetails.productCode)  (cost=0.74 rows=1) (actual time=0.000..0.000 rows=1 loops=2996)
```

Results discussion

The output of EXPLAIN ANALYZE after adding the new index shows a reduced cost and actual execution time, indicating that the query is more efficient. The query plan shows that the query is using the new index "idx_orderdetails_productCodeNo" that was created earlier, resulting in a covering index scan on the orderdetails table.
Therefore, the new index has improved the performance of the query.