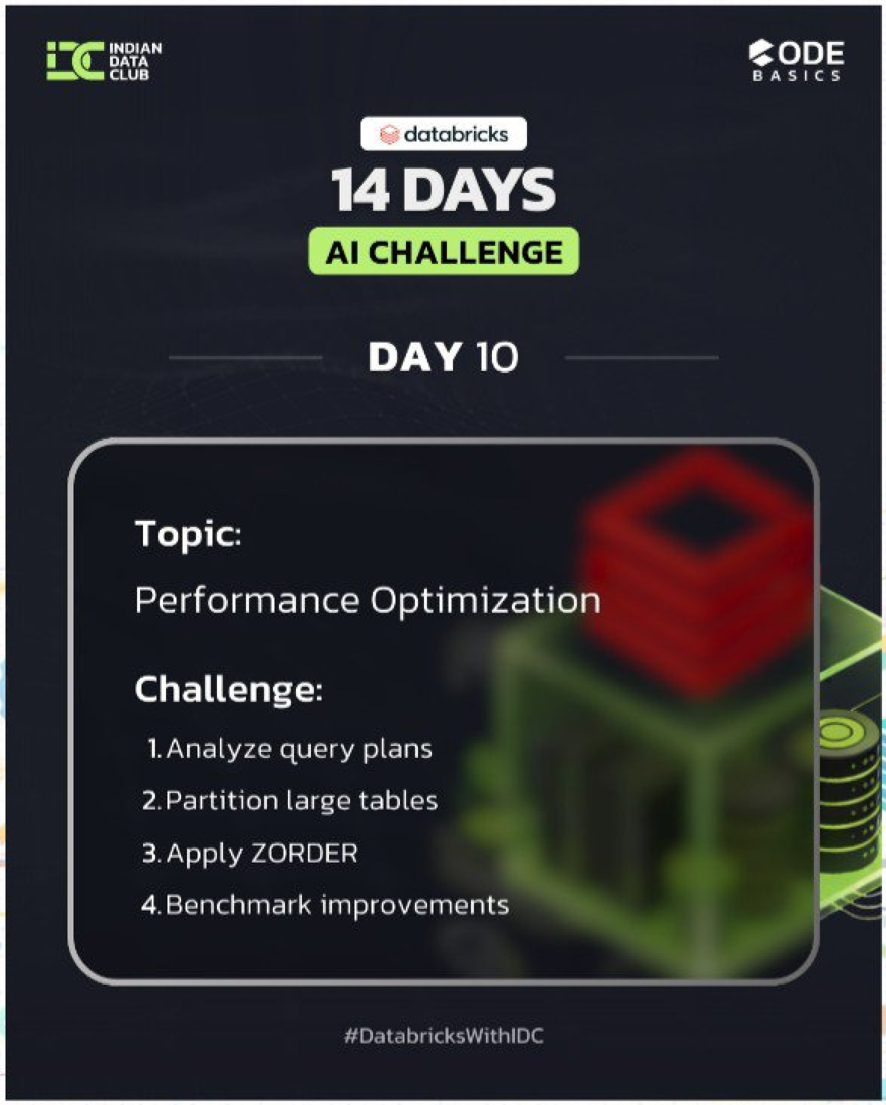
DAY 10 – Performance Optimization



INDIAN DATA CLUB

CODE BASICS

databricks

14 DAYS

AI CHALLENGE

DAY 10

Topic:

Performance Optimization

Challenge:

1. Analyze query plans
2. Partition large tables
3. Apply ZORDER
4. Benchmark improvements

#DatabricksWithIDC

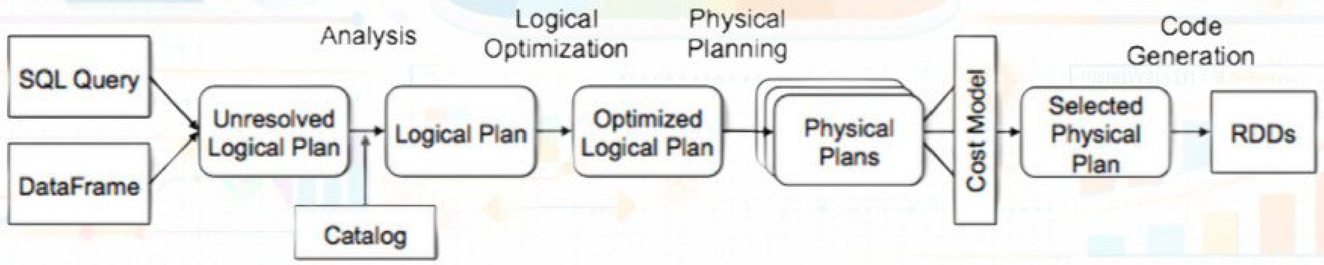
What is a Query Execution Plan?

◆ *A query execution plan shows how Spark runs your query internally*

◆ *Includes:*

- *Table scans*
- *Filters*
- *Joins*
- *Aggregations*
- *Shuffles*

◆ *Helps identify performance bottlenecks*



Why Query Plans Matter?

- *Reveals why a query is slow*
- *Helps detect:*

•Full table scans
•Expensive shuffles
•Inefficient joins

- *Essential for query tuning & optimization*

How to Analyze Query Plans

SQL :

```
EXPLAIN FORMATTED SELECT * FROM
table;
```

PySpark:

```
df.explain(True)
```

Key things to look for:
•Shuffle operations
•Filter pushdown
•Join strategies

What is Partitioning?

◆ *Partitioning splits a table into folders based on a column*

◆ *Example:*

- *order_date=2024-01-01*
- *order_date=2024-01-02*

◆ *Reduces data scanned during queries*

Benefits of Partitioning

- *Faster query performance*
- *Lower compute cost*
- *Efficient filtering on partition columns*
- *Improves scalability for large datasets*

Partitioning Best Practices

- *Use low to medium cardinality columns*
- *Best columns:*

•Date, year, month

- *Avoid:*

•High-cardinality columns (customer_id)

- *Don't over-partition (small files problem)*

OPTIMIZE in Databricks

•OPTIMIZE compacts many small files into fewer large files
•Improves read performance
•Reduces file management overhead

```
OPTIMIZE sales_table;
```

What is ZORDER?

•ZORDER clusters related data inside files
•Improves query performance for selective filters
•Works best on high-cardinality columns

```
OPTIMIZE sales_table
ZORDER BY (customer_id);
```

How to Cache Data

SQL :

```
CACHE TABLE sales_table;
```

PySpark:

```
df.cache()
df.count()
```

Uncache when done :

```
UNCACHE TABLE sales_table;
```

When to Use Caching

✅ *Use caching for:*

- *Dashboards*
- *Reused tables*
- *Intermediate datasets*

❌ *Avoid caching:*

- *One-time queries*
- *Very large tables*
- *Memory-intensive workloads*