## 🧠 Dynamic Partition Pruning (DPP

### What is Partition Pruning?

In partitioned tables, data is split into **subdirectories** (partitions) based on partition keys (e.g., date, region).
**Partition Pruning** means reading **only the partitions needed** for a query instead of scanning the entire dataset.
This reduces I/O and speeds up queries.

Static vs Dynamic Partition Pruning

| Feature | Static Partition Pruning | Dynamic Partition Pruning |
|---|---|---|
| Partition filter value | Known at **query compile time** | Determined **at runtime** |
| Example | WHERE region = 'US' | WHERE region IN (SELECT region FROM other_table) |
| When used | Simple filters | Filters depend on another query result or runtime parameter |
| Performance | Very efficient | Efficient but slightly more overhead due to runtime evaluation |

### Why Dynamic Partition Pruning is Needed

Many queries have partition filter values **not known in advance**.

```
SELECT *  FROM sales WHERE region IN (   SELECT region   FROM active_regions   WHERE year = 2024 );
```

The region values come from another table.
Without **DPP**, all partitions of sales would be scanned → slow and expensive.
With **DPP**, only partitions matching the runtime region values are read.

### How DPP Works in Spark

**Static PP** happens at compile/optimization time.
**Dynamic PP**:

1. Spark compiles the query with a placeholder for the partition filter.
2. At runtime, executes the subquery to get filter values.
3. Applies partition pruning before scanning the table.

```
SET spark.sql.optimizer.dynamicPartitionPruning.enabled=true;
SELECT * FROM fact_sales WHERE date_id IN (
   SELECT DISTINCT date_id      FROM dim_calendar      WHERE holiday_flag = 'Y' );
```

Spark Configs to Enable

**SET spark.sql.optimizer.dynamicPartitionPruning.enabled = true;**

This tells Spark to actually perform runtime partition pruning.

## Visual Example

Imagine your dataset has partitions:
/sales/region=US/
/sales/region=CA/
/sales/region=UK/
/sales/region=IN/

Query asks for:
SELECT * FROM sales WHERE region IN (SELECT region FROM active_regions);

Without DPP: Scans all 4 partitions.

With DPP: Scans only the partitions returned by active_regions (maybe just US and CA).

## How DPP Works in Spark

1. Compilation Phase
    - Spark creates a plan with a placeholder for the partition filter.
    - Knows it will get the filter values from another query stage.
2. Runtime
    - Executes the filter-producing subquery first.
    - Obtains actual values (e.g., ['US', 'CA']).
    - Applies these values to the scan of the partitioned table.
3. Scan
    - Reads only the relevant partitions (e.g., /region=US/ and /region=CA/).
    - Skips all others.

## Visual Example

Imagine your dataset has partitions:
/sales/region=US/
/sales/region=CA/
/sales/region=UK/
/sales/region=IN/

```
SET spark.sql.optimizer.dynamicPartitionPruning.enabled = true;
SELECT * FROM sales WHERE region IN (  SELECT region  FROM active_regions  WHERE year = 2025);
```

## Without DPP

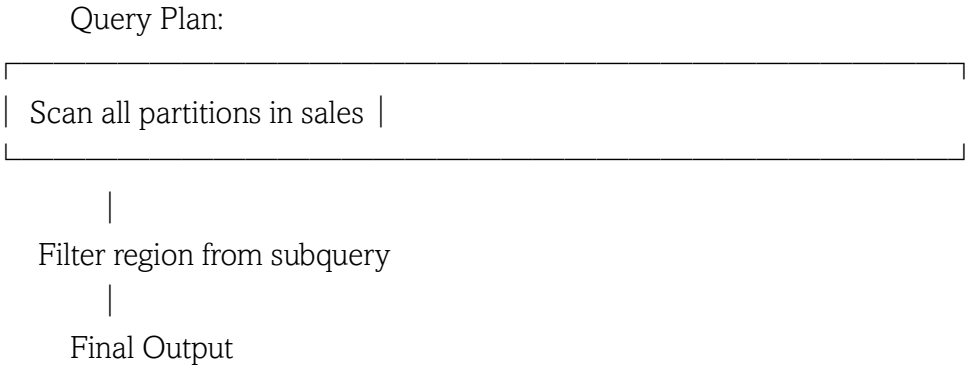
- Spark reads all four partitions.
- Filters out unwanted rows after reading.

## With DPP

- Spark first runs SELECT region FROM active_regions WHERE year=2025
- Suppose result = ['US', 'CA']
- Spark reads only /region=US/ and /region=CA/.

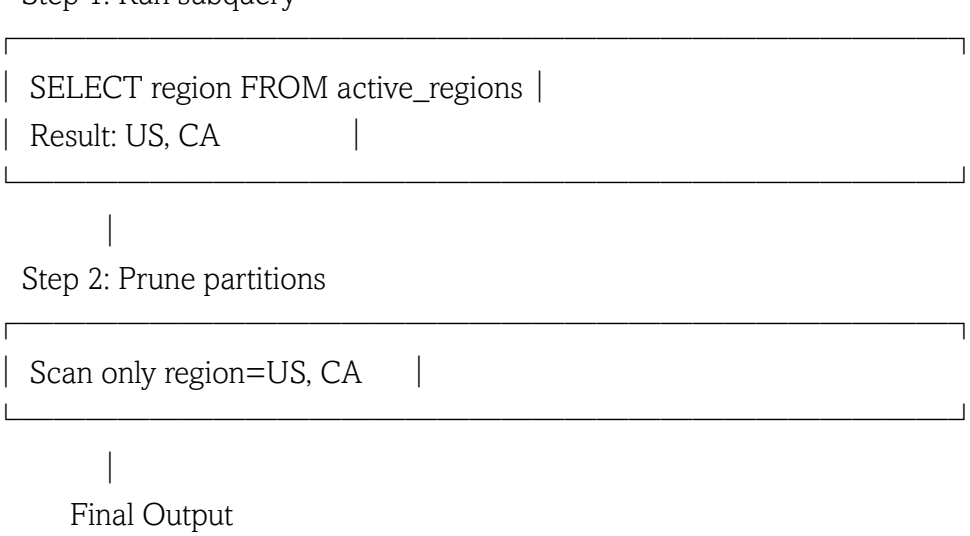
## Before vs After Diagram

Before DPP (Full Scan)




Reads **US, CA, UK, IN** partitions → **Slow**.

After DPP (Pruned Scan)




Reads **only needed partitions** → **Fast**.

## Benefits

- **Performance**: Avoids unnecessary I/O.
- **Cost Savings**: In cloud environments (S3, GCS, ADLS), reading less data = lower cost.
- **Scalability**: Handles large datasets efficiently.

## Spark Configs for DPP

```
-- Enable Dynamic Partition Pruning
SET spark.sql.optimizer.dynamicPartitionPruning.enabled = true;

-- Optional: Increase timeout if filter value fetch takes longer
SET spark.sql.dynamicPartitionPruning.broadcast.timeout = 5m;
```

## Key Takeaways

- Use DPP when your filter values come from another dataset or runtime parameter.
- Ensure your table is partitioned **on the filter column** for maximum effect.
- Validate in **Spark UI** by checking the number of partitions scanned.