04 Hive Partitioning-KirkYagami-NikhilSharma👨‍💻🕵️

Hive Partitioning

Suggested Readings: https://sparkbyexamples.com/apache-hive/hive-partitions-explained-with-examples/
Another Topic, Bucketing in Hive: https://sparkbyexamples.com/apache-hive/hive-bucketing-explained-with-examples/

Partitioning in Hive is a powerful feature used to optimize data storage and improve query performance for large datasets. By breaking down a large table into smaller partitions based on column values, Hive reduces the amount of data scanned during queries. In this lecture, we will explore the details of Hive partitioning, from creating partitioned tables to best practices, fine-tuning, and dynamic partitioning.

---

1. What is Partitioning in Hive?

Partitioning is a way to divide a table into smaller parts, called partitions, based on the values of one or more columns. These partitions are stored as separate subdirectories in HDFS. Each partition can contain one or more files, making it easier to manage large datasets and optimize query performance.

- **Partition Columns**: Columns used to split the table's data into partitions.
- **Partitions as Directories**: In HDFS, each partition is a separate directory containing data related to that partition.

Example Scenario:

Imagine you have a large sales table with millions of rows, and most of your queries filter by `OrderDate`. Partitioning the table by `OrderDate` will allow you to scan only the relevant partitions, significantly reducing the amount of data processed during queries.

---

2. Benefits of Partitioning

1. **Improved Query Performance**: When a query includes filters on partitioned columns, Hive can skip unnecessary partitions, reducing scan times and improving query performance.

2. **Efficient Data Management**: Partitioning helps organize data into smaller, more manageable parts, making it easier to handle and process large datasets.
3. **Optimized Storage**: Partitioning organizes data into subdirectories in HDFS, leading to more efficient storage and retrieval.
4. **Scalability**: Partitioned tables can scale well for big data systems by distributing the load more evenly across partitions.

---

3. Best Practices for Partitioning

- **Choose the Right Partition Columns**: Select columns that are commonly used in `WHERE` clauses. For time-series data, partitioning by date is a common approach.
- **Avoid Too Many Partitions**: Excessive partitioning can overwhelm the Hadoop NameNode, which stores metadata for all partitions. A large number of small partitions can degrade performance.
- **Balance Between Partition Size and Number of Partitions**: Ensure partitions are neither too small nor too large. Small partitions lead to too many directories, while large partitions negate the benefits of partitioning.

---

4. Creating Tables with Partitions

Example 1: Partitioning by One Column

Let's create a table partitioned by `OrderDate`.

```
CREATE TABLE Sales_Data_Partitioned_By_Date
(
    StoreLocation STRING,
    Product STRING,
    Revenue DECIMAL(10,2)
)
PARTITIONED BY (OrderDate DATE);
```

- This table is partitioned by `OrderDate`. Each distinct `OrderDate` will create a separate partition in HDFS.

Example 2: Partitioning by Two Columns

You can also partition by multiple columns. Let's add `StoreLocation` as a second partition column.

```
CREATE TABLE Sales_Data_Partitioned_By_Location_And_Date
(
    Product STRING,
    Revenue DECIMAL(10,2)
)
PARTITIONED BY (StoreLocation STRING, OrderDate DATE);
```

- This table is partitioned by both `StoreLocation` and `OrderDate`. Partitions are created for each combination of these two columns.

---

5. HDFS File Structure of Partitioned Tables

When you partition a Hive table, Hive organizes the data into subdirectories in HDFS. Each partition is stored in a separate subdirectory under the table's directory.

Example:

For the table `Sales_Data_Partitioned_By_Location_And_Date` partitioned by `StoreLocation` and `OrderDate`, Hive creates directories like:

```
/user/hive/warehouse/sales_data_partitioned_by_location_and_date/
├── StoreLocation=Bellandur
│   ├── OrderDate=2023-01-18/
│   ├── OrderDate=2023-01-19/
├── StoreLocation=Koramangala
│   ├── OrderDate=2023-01-18/
│   ├── OrderDate=2023-01-19/
```

Each partition directory contains files with the data for that partition.

---

6. How Queries Execute on Partitioned Tables

When you run queries on partitioned tables, Hive optimizes the query by scanning only the relevant partitions based on the `WHERE` clause filters.

Example Query:

```
SELECT SUM(Revenue)
FROM Sales_Data_Partitioned_By_Date
```

```
WHERE OrderDate = '2023-01-18';
```

Hive will scan only the partition `OrderDate=2023-01-18`, ignoring all other partitions. This dramatically improves query performance compared to a non-partitioned table, where the entire table would need to be scanned.

---

7. Dynamic Partitioning

Dynamic partitioning allows Hive to automatically create partitions based on the data being inserted, without manually specifying the partition values. This is useful when dealing with large datasets and multiple partitions.

Enabling Dynamic Partitioning:

Before inserting data with dynamic partitioning, you must enable it in Hive.

```
SET hive.exec.dynamic.partition = true;
SET hive.exec.dynamic.partition.mode = nonstrict;
```

- `hive.exec.dynamic.partition`: Enables dynamic partitioning.
- `hive.exec.dynamic.partition.mode`: When set to `nonstrict`, all partitions can be dynamic. In `strict` mode, at least one partition must be static.

Dynamic Partition Insertion Example:

```
INSERT INTO TABLE Sales_Data_Partitioned_By_Date
PARTITION (OrderDate)
VALUES
('Bellandur', 'Bananas', 8236.33, '2023-01-18'),
('Koramangala', 'Nutella', 7455.67, '2023-01-18');
```

Hive automatically creates the necessary partitions (`OrderDate=2023-01-18`) and stores the data accordingly.

---

8. Strict Mode vs Non-Strict Mode

Hive has two partition modes: **Strict** and **Non-Strict**.

- **Strict Mode**: At least one partition key must be specified as a static value in the query. Dynamic partitioning is allowed only if there is at least one static partition column.

Example: You cannot leave all partition columns dynamic.
- **Non-Strict Mode**: All partition columns can be dynamic, meaning partitions can be automatically created without requiring static values.

---

9. How to Check What Partitions Exist in a Table

You can list all the existing partitions in a Hive table using the following command:

```
SHOW PARTITIONS Sales_Data_Partitioned_By_Date;
```

This will return a list of all partitions for the table.

Example Output:

```
OrderDate=2023-01-17
OrderDate=2023-01-18
```

Conclusion

Hive partitioning is an essential technique for optimizing performance when working with large datasets. By dividing tables into partitions, Hive reduces the amount of data scanned during queries, leading to faster query execution and better resource utilization.

- **Best Practices**: Always choose partition columns based on query patterns, avoid over-partitioning, and enable dynamic partitioning for efficient data management.
- **Fine-Tuning**: Use the correct partitioning strategies to balance performance and manageability, and configure dynamic partitioning based on your specific use case.

StoreLocation	Product	Date	Revenue
Bellandur	Nutella	January 18,2016	7,455.67
Bellandur	Peanut Butter	January 18,2016	5,316.89
Bellandur	Milk	January 18,2016	2,433.76
Koramangala	Bananas	January 18,2016	9,456.01
Koramangala	Nutella	January 18,2016	3,644.33
Koramangala	Peanut Butter	January 18,2016	8,988.64
Koramangala	Milk	January 18,2016	1,621.58

CONSIDER THE SALES TABLE

YOU CAN USE ANY COLUMN TO CREATE PARTITIONS

StoreLocation	Product	Date	Revenue
Bellandur	Nutella	January 18,2016	7,455.67
Bellandur	Peanut Butter	January 18,2016	5,316.89
Bellandur	Milk	January 18,2016	2,433.76
Koramangala	Bananas	January 18,2016	9,456.01
Koramangala	Nutella	January 18,2016	3,644.33
Koramangala	Peanut Butter	January 18,2016	8,988.64
Koramangala	Milk	January 18,2016	1,621.58

CONSIDER THE SALES TABLE

LET'S PARTITION IT ON THE PRODUCT COLUMN

YOU CAN USE ANY COLUMN TO CREATE PARTITIONS

StoreLocation	Product	Date	Revenue
Bellandur	Nutella	January 18,2016	7,455.67
Bellandur	Peanut Butter	January 18,2016	5,316.89
Bellandur	Milk	January 18,2016	2,433.76
Koramangala	Bananas	January 18,2016	9,456.01
Koramangala	Nutella	January 18,2016	3,644.33
Koramangala	Peanut Butter	January 18,2016	8,988.64
Koramangala	Milk	January 18,2016	1,621.58

LET'S PARTITION IT ON THE PRODUCT COLUMN

THERE ARE 4 DISTINCT VALUES IN PRODUCT COLUMN

Product
Bananas
Nutella
Peanut Butter
Milk

$\rightarrow$ 4 PARTITIONS OF THIS TABLE

LET'S PARTITION IT ON THE PRODUCT COLUMN

StoreLocation	Product	Date	Revenue
Bellandur	Bananas	January 18,2016	8,236.33
Bellandur	Nutella	January 18,2016	7,455.67
Bellandur	Peanut Butter	January 18,2016	5,316.89
Bellandur	Milk	January 18,2016	2,433.76
Koramangala	Bananas	January 18,2016	9,456.01
Koramangala	Nutella	January 18,2016	3,644.33
Koramangala	Peanut Butter	January 18,2016	8,988.64
Koramangala	Milk	January 18,2016	1,621.58

4 PARTITIONS OF THIS TABLE

PRODUCT = BANANAS

StoreLocation	Date	Revenue
Bellandur	January 18,2016	8,236.33
Koramangala	January 18,2016	9,456.01

PRODUCT = PEANUT BUTTER

StoreLocation	Date	Revenue
Bellandur	January 18,2016	5,316.89
Koramangala	January 18,2016	8,988.64

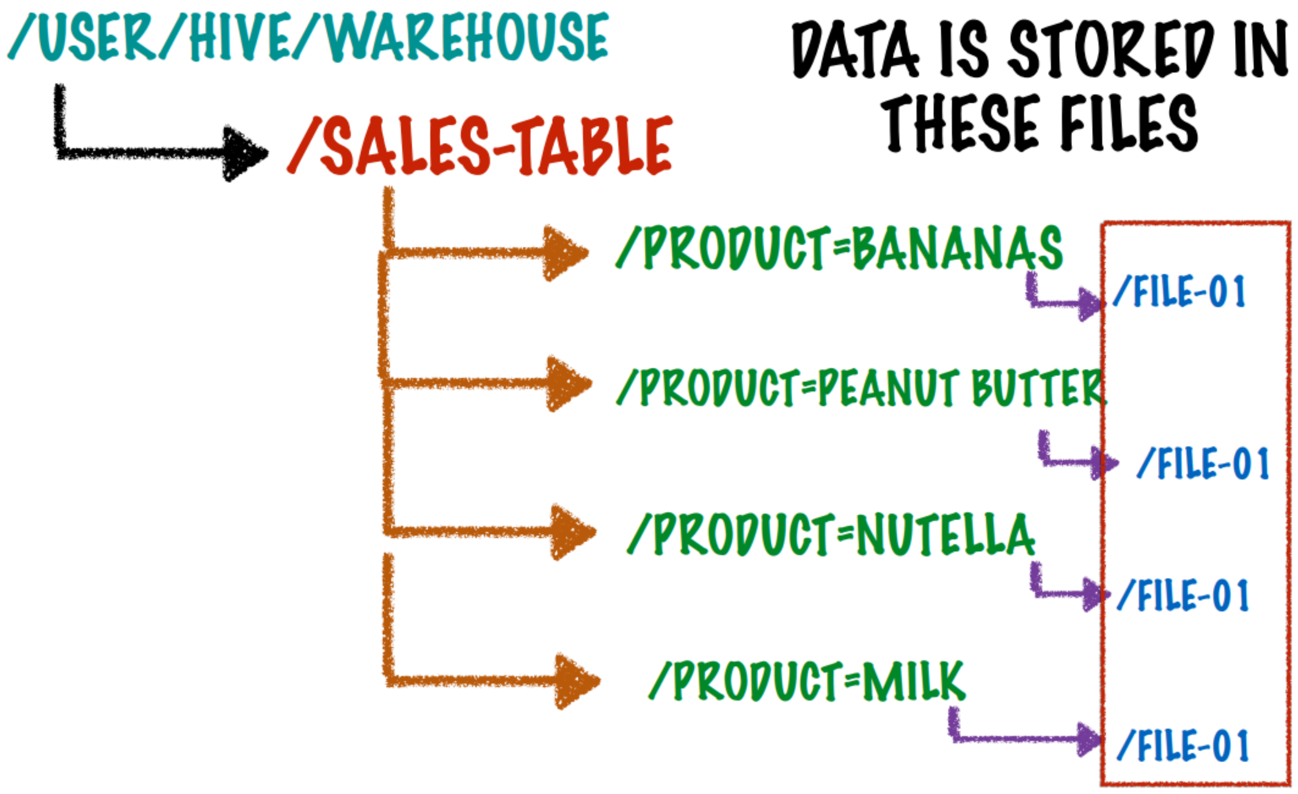
PRODUCT = NUTELLA

StoreLocation	Date	Revenue
Bellandur	January 18,2016	7455.67
Koramangala	January 18,2016	3644.33

PRODUCT = MILK

StoreLocation	Date	Revenue
Bellandur	January 18,2016	2,433.76
Koramangala	January 18,2016	1,621.58

THESE BECOME 4 SUB-DIRECTORIES IN THE TABLE'S DIRECTORY



PARTITIONING IMPROVES QUERY PERFORMANCE

CALCULATE TOTAL REVENUE FROM SELLING MILK ON JANUARY 17

THIS IS HOW OUR PARTITIONED TABLE LOOKS

PRODUCT = BANANAS

StoreLocation	Date	Revenue
Bellandur	January 18,2016	8,236.33
Koramangala	January 18,2016	9,456.01

PRODUCT = PEANUT BUTTER

StoreLocation	Date	Revenue
Bellandur	January 18,2016	5,316.89
Koramangala	January 18,2016	8,988.64

PRODUCT = NUTELLA

StoreLocation	Date	Revenue
Bellandur	January 18,2016	7455.67
Koramangala	January 18,2016	3644.33

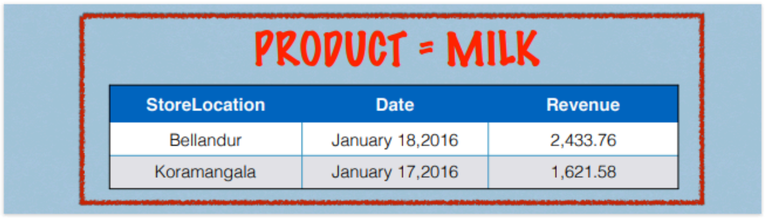
PRODUCT = MILK

StoreLocation	Date	Revenue
Bellandur	January 18,2016	2,433.76
Koramangala	January 17,2016	1,621.58

PARTITIONING IMPROVES QUERY PERFORMANCE

CALCULATE TOTAL REVENUE FROM SELLING MILK ON JANUARY 17

THIS IS HOW OUR PARTITIONED TABLE LOOKS



PRODUCT = MILK

StoreLocation	Date	Revenue
Bellandur	January 18,2016	2,433.76
Koramangala	January 17,2016	1,621.58

IN A PARTITIONED TABLE, WE JUST SCAN THE PARTITION CORRESPONDING TO PRODUCT = MILK

HOW TO PARTITION TABLES?

BY ADDING

```
partitioned by(Partition_Column_Name column_data_type)
```

```
CREATE TABLE Sales_Data_Product_Partition
(
StoreLocation VARCHAR(30),
OrderDate DATE,
Revenue DECIMAL(10,2)
)
partitioned by(product varchar(30));
```

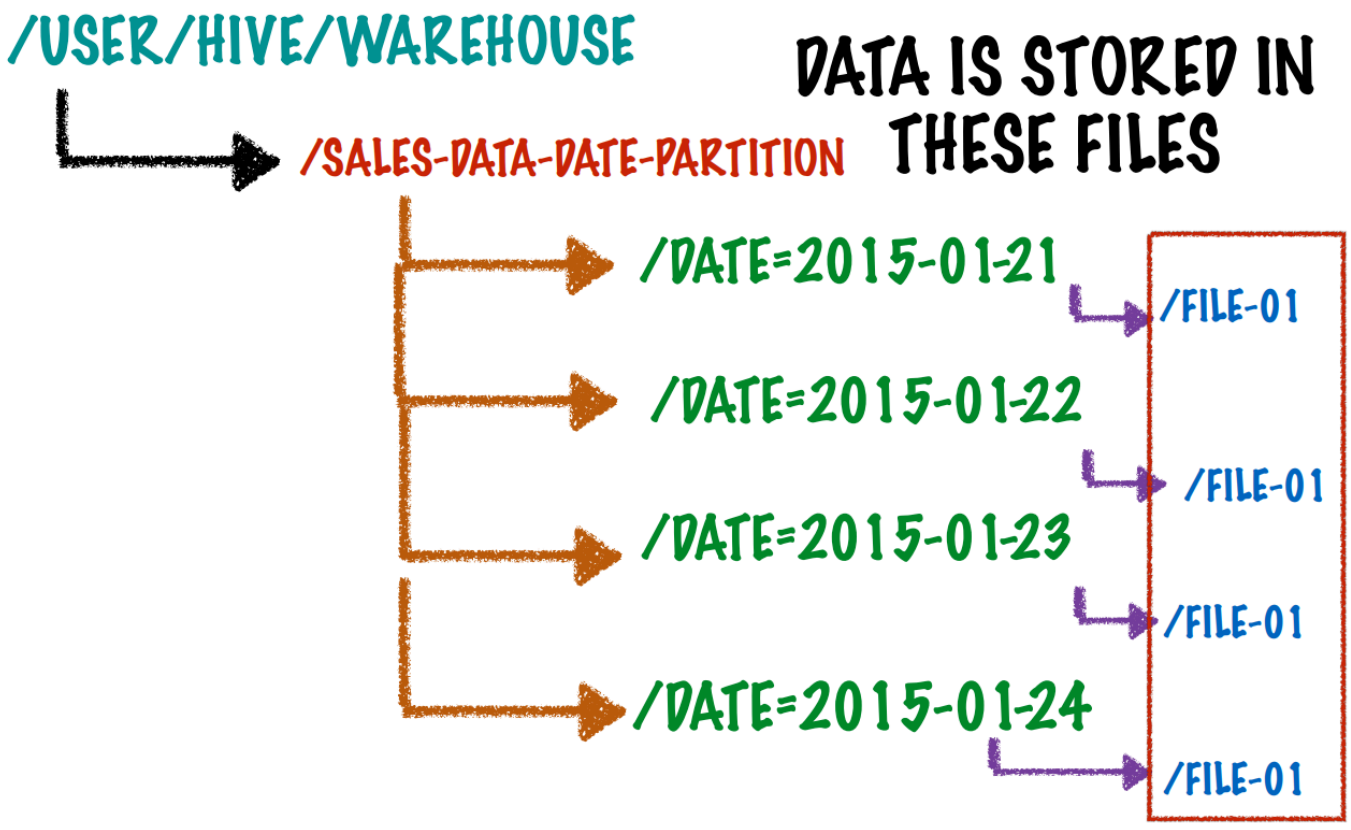
> NOTE THAT THE COLUMN IS NOT SPECIFIED AS A PART OF THE CREATE TABLE

IF WE WANT TO PARTITION TABLES BY DATE COLUMN

BY ADDING

```
partitioned by(column name column_data_type)
```

```
CREATE TABLE Sales_Data_Date_Partition
(
StoreLocation VARCHAR(30),
product VarChar(30),
Revenue DECIMAL(10,2)
)
partitioned by(OrderDate DATE);
```



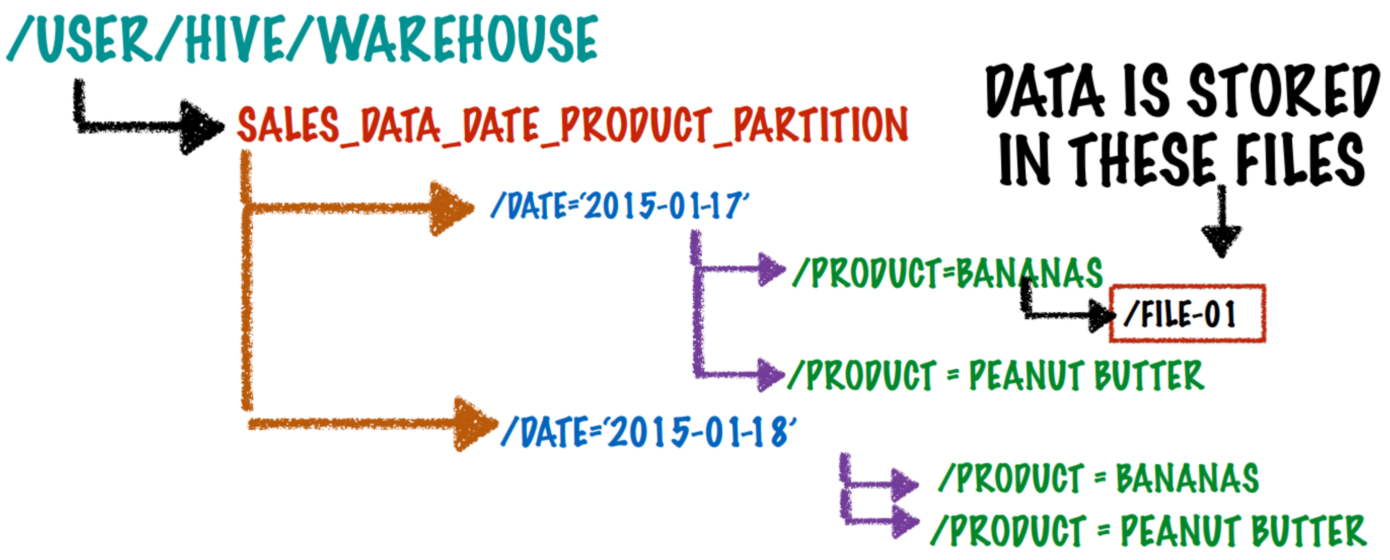
HOW DO WE GET DATA FROM TABLES WITH PARTITIONS?

```
CREATE TABLE Sales_Data_Date_Product_Partition
(
StoreLocation VARCHAR(30),
Revenue DECIMAL(10,2)
)
partitioned by
(
OrderDate DATE,
product VarChar(30)
);
```

WHEN YOU QUERY THIS TABLE JUST TREAT IT AS IF IT HAS 4 COLUMNS

WE CAN PARTITION TABLES BY TWO COLUMNS

LET'S LOOK AT THE DIRECTORY STRUCTURE FOR SUCH A TABLE


/USER/HIVE/WAREHOUSE
SALES_DATA_DATE_PRODUCT_PARTITION
DATA IS STORED IN THESE FILES
/DATE='2015-01-17'
/PRODUCT=BANANAS
/FILE-01
/PRODUCT = PEANUT BUTTER
/DATE='2015-01-18'
/PRODUCT = BANANAS
/PRODUCT = PEANUT BUTTER