Working with Spark SQL, UDFs & Common DataFrame Operations

Mohit Batra
Founder, Crystal Talks

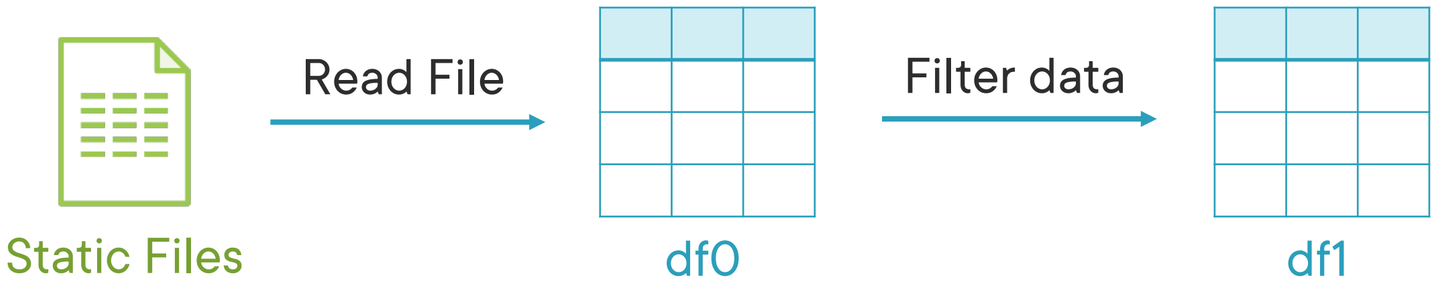
linkedin.com/in/mohitbatra

Overview

- Run SQL queries on DataFrames
- Work with Spark databases & tables
- Work with user-defined functions (UDFs)
- Perform operations on multiple datasets
- Perform window operations

Running SQL Queries on DataFrames

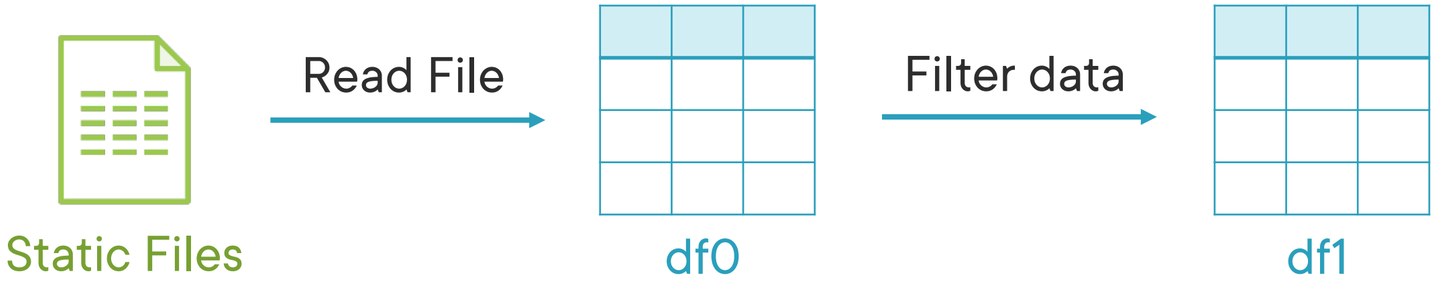
PySpark / Scala




```
df0 = spark
        .read
        .load (...)
```

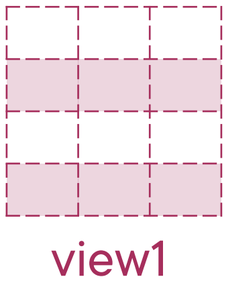
```
df1 = df0
        .where (...)
```

PySpark / Scala



Create Temp View
on DataFrame

SQL




```
df1.createOrReplaceTempView("view1")
```

```
spark.sql( "SELECT * FROM view1" )
```

SQL Queries on DataFrames

Part of Spark SQL library

Create SQL View on top of DataFrame

- *Think of this like pointer to DataFrame*

Run SQL queries on the View

Same performance in SQL as using DataFrames in other languages

Use spark.sql in PySpark/Scala to run SQL queries

spark.sql returns output of query as a DataFrame

Working with Spark Tables

Writing Data from DataFrame

Directly to Storage

Files stored in defined format in Storage

No metadata registered

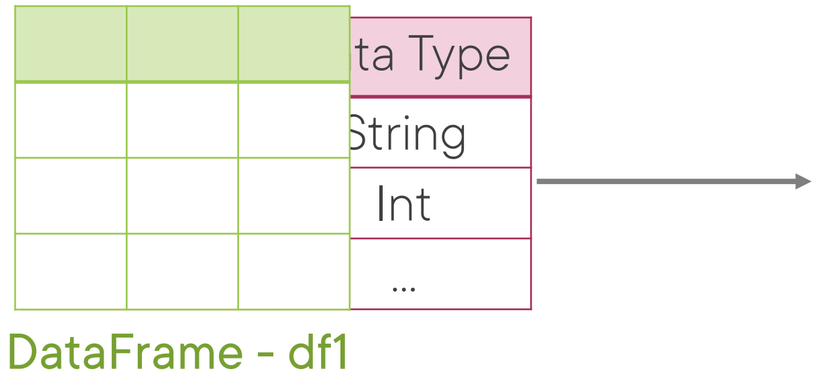
To query, read files from Storage

As Spark Table

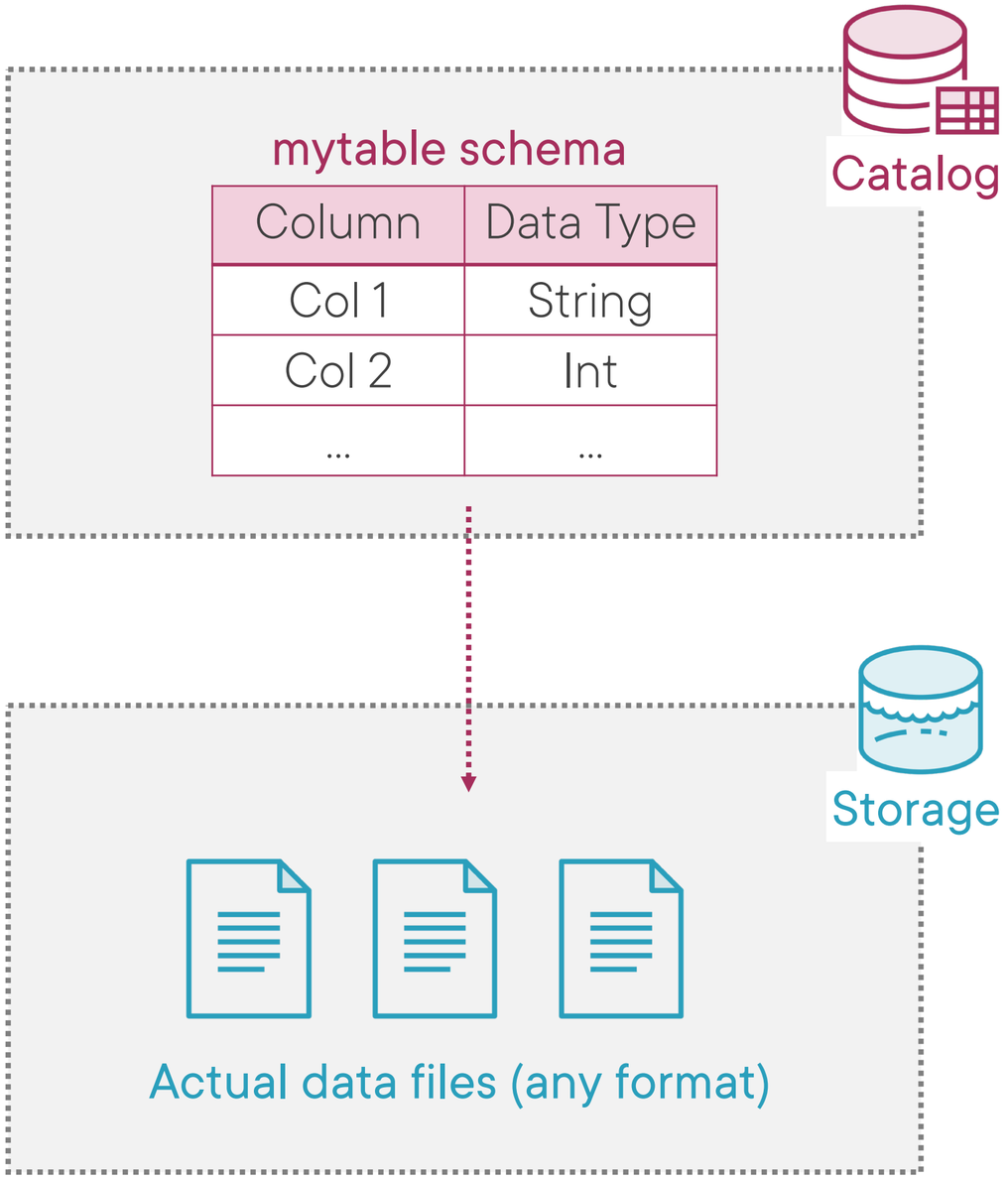
Files stored in defined format in Storage

Metadata registered with a metastore

To query, either read files from Storage or directly reference the table




```
df1
    .write
    .saveAsTable (“mytable”)
```

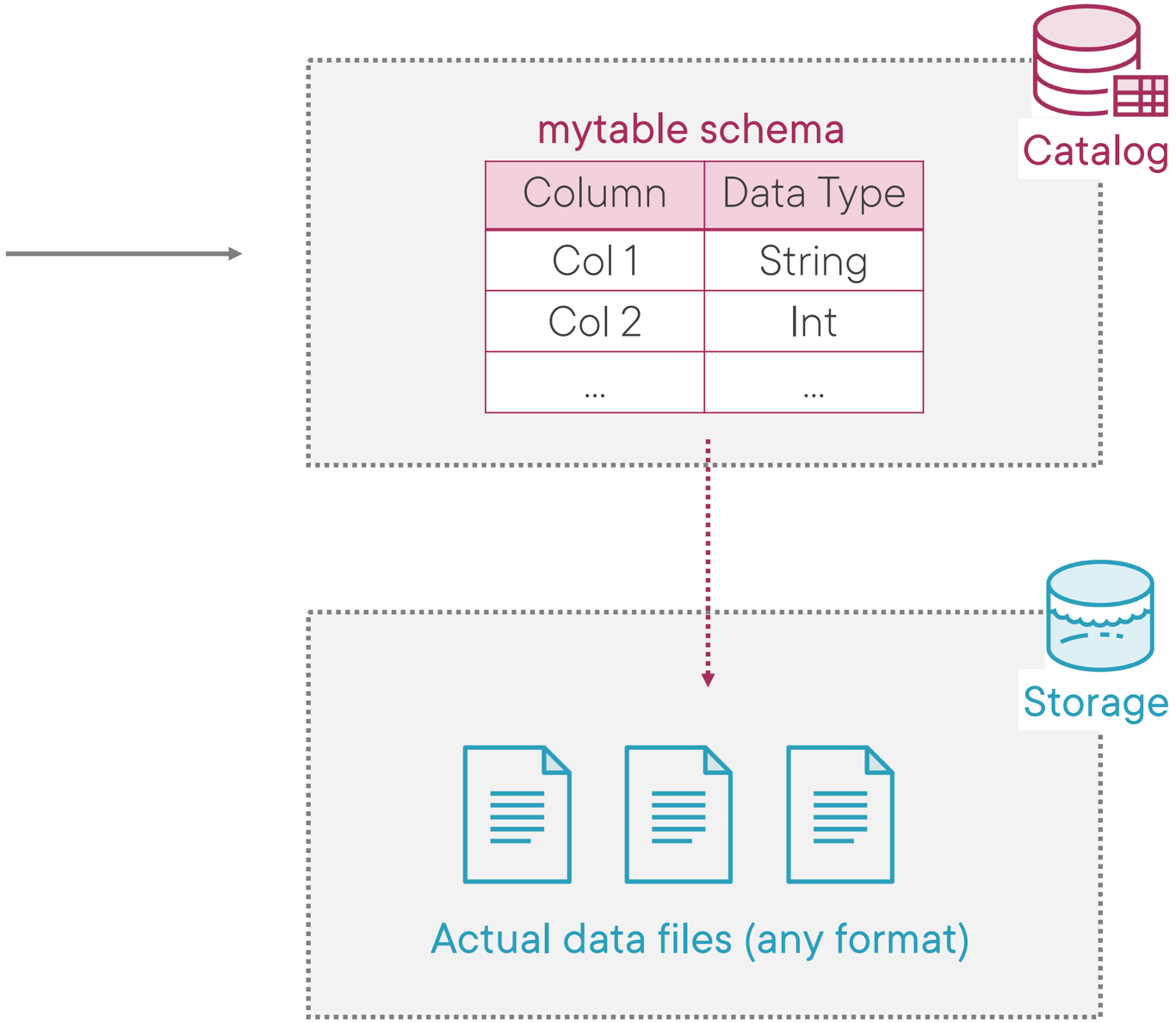


SQL

```
SELECT * FROM mytable
```

PySpark

```
df1 = spark
        .read
        .table("mytable")
```



Helps to manage datasets

No need to define/infer schema while reading

Catalog Types

In-memory Catalog

Only works in a session
Cleaned up when session ends

Persistent Catalog

Metadata is permanently stored
Can be used in any session
Spark has built-in **Hive** catalog

Types of Tables

Managed Table

Schema and data is managed by Spark

Data is stored in default location

Dropping table deletes both schema & data

Useful to persist staging data

Unmanaged / External Table

Only schema is managed by Spark

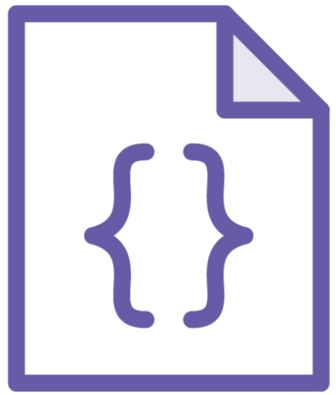
Data is stored in an external location

Dropping table deletes only schema, not data

Useful to persist processed data

Working with User Defined Functions (UDFs)

User Defined Functions



Extends functionality with custom Scala/Python methods

Invoked for each row in DataFrame

To register a function as Spark UDF:

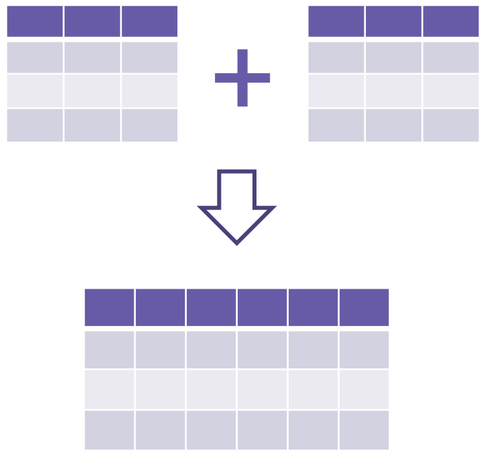
- Register using udf() to use with DataFrames
- Register using spark.udf.register() to use in SQL

Challenges

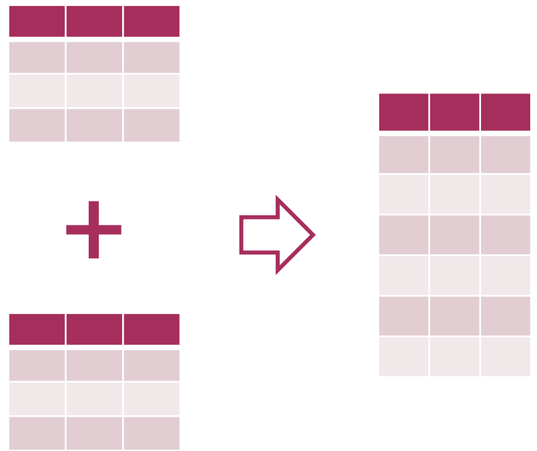
- Optimizations can't be applied to UDFs
- Nulls can cause issues if not handled well

Performing Operations on Multiple Datasets

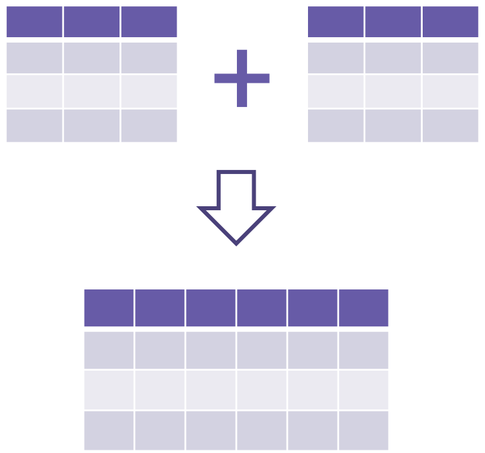
Operations on Multiple Datasets



Join Operations



Set Operations



Join Operations

Combines two datasets, based on conditions

Produces result with new set of columns

Write code in Python/Scala or use SQL queries

Spark supports

- Inner Join
- Left Outer Join
- Right Outer Join
- Full Outer Join
- Cross Join *(Python/Scala only)*
- Semi Join *(Python/Scala only)*
- Anti Join *(Python/Scala only)*

Spark 3 has several optimizations for Join operations

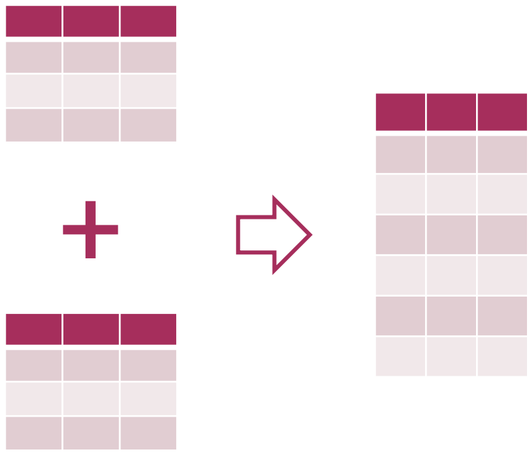
Inner Join

Dept Id	Dept Name
D1	Sales
D2	Marketing
D3	Tech

On
Dept Id

Dept Id	Emp Id	Emp Name
D1	E1	Anna
D1	E2	Steve
D3	E3	Neha
D5	E4	Ivan

Dept Id	Dept Name	Dept Id	Emp Id	Emp Name
D1	Sales	D1	E1	Anna
D1	Sales	D1	E2	Steve
D3	Tech	D3	E3	Neha



Set Operations

Applies to two datasets having same schema, without conditions

Output dataset has same schema as input datasets

Write code in Python/Scala or use SQL queries

Spark supports

- Union
- Union All
- Intersect
- Except / Minus

Union All

User Id	Name
U1	Anna
U2	Cristina
U3	Neha
U4	Steve

User Id	Name
U4	Steve
U5	Ivan
U6	Mohit

User Id	Name
U1	Anna
U2	Cristina
U3	Neha
U4	Steve
U4	Steve
U5	Ivan
U6	Mohit

Union

User Id	Name
U1	Anna
U2	Cristina
U3	Neha
U4	Steve

User Id	Name
U4	Steve
U5	Ivan
U6	Mohit

User Id	Name
U1	Anna
U2	Cristina
U3	Neha
U4	Steve
U5	Ivan
U6	Mohit

Intersect

User Id	Name
U1	Anna
U2	Cristina
U3	Neha
U4	Steve

User Id	Name
U4	Steve
U5	Ivan
U6	Mohit

User Id	Name
U4	Steve

Except / Minus

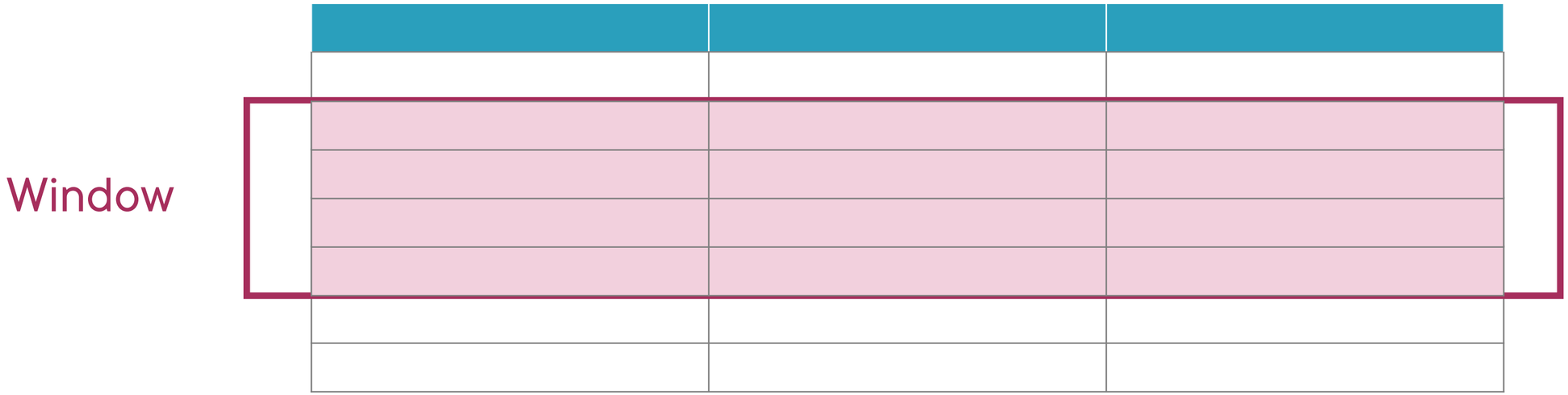
User Id	Name
U1	Anna
U2	Cristina
U3	Neha
U4	Steve

User Id	Name
U4	Steve
U5	Ivan
U6	Mohit

User Id	Name
U1	Anna
U2	Cristina
U3	Neha

Performing Window Operations

Window



Window is a subset of related rows in a DataFrame

Define how rows are related, say by time or location

Apply operations

- count, sum, average, min, max etc.

For each row in Window, operation is applied on all the rows of Window

Find share of each department in total expenses

Department	Expense
HR	10,000
Tech	20,000
Sales	30,000
Marketing	40,000

Window is set of rows from Departments table

Find share of each department in total expenses

Department	Expense
HR	10,000
Tech	20,000
Sales	30,000
Marketing	40,000

Expense % = Expense * 100 / Total Expenses

Find share of each department in total expenses

Department	Expense	Total Expenses
HR	10,000	100,000
Tech	20,000	100,000
Sales	30,000	100,000
Marketing	40,000	100,000

Expense % = Expense * 100 / Total Expenses

Find share of each department in total expenses

Department	Expense	Total Expenses	Expense %
HR	10,000	100,000	10
Tech	20,000	100,000	20
Sales	30,000	100,000	30
Marketing	40,000	100,000	40

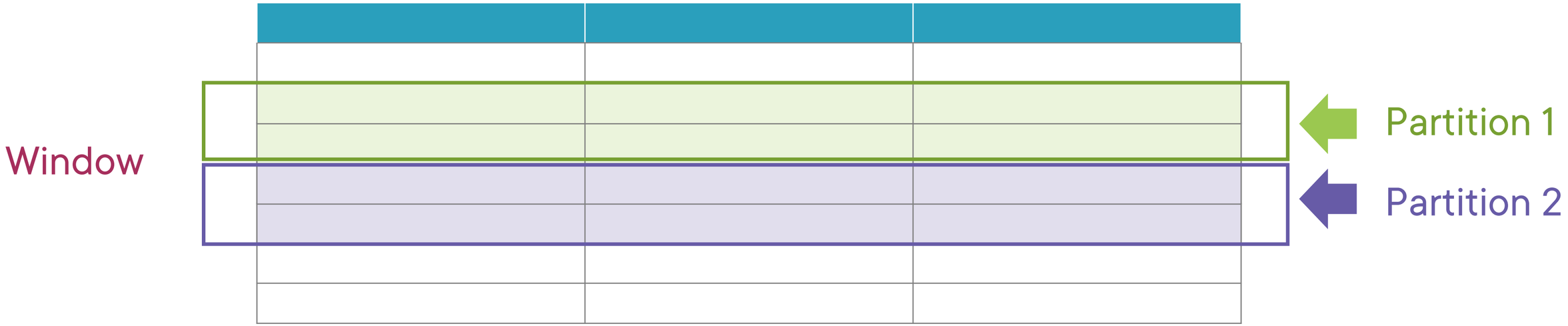
Expense % = Expense * 100 / Total Expenses

Find share of each department in total expenses

Department	Expense	Total Expenses	Expense %
HR	10,000	100,000	10
Tech	20,000	100,000	20
Sales	30,000	100,000	30
Marketing	40,000	100,000	40

```
SELECT *
        , Expense * 100 / TotalExpenses        AS ExpensePercent
FROM (
            SELECT Department, Expense
                    , SUM (Expense)    OVER ()      AS TotalExpenses
            FROM Departments
        ) subquery
```

Window Partitions



Window can be further divided into Partitions

Operations are then applied on Partitions

- count, sum, average, min, max etc.

Find share of each employee in their department's payout

Department	Employee	Salary
Sales	Anna	8,000
Marketing	Steve	9,500
Tech	Cristina	9,000
Marketing	Neha	10,500
Tech	Kari	10,000
Sales	Ivan	10,000
Tech	Mohit	8,000

Window is set of rows from Employees table

Find share of each employee in their department's payout

Department	Employee	Salary
Sales	Anna	8,000
Marketing	Steve	9,500
Tech	Cristina	9,000
Marketing	Neha	10,500
Tech	Kari	10,000
Sales	Ivan	10,000
Tech	Mohit	8,000

Partition by Department

Find share of each employee in their department's payout

Department	Employee	Salary
Sales	Anna	8,000
Sales	Ivan	10,000
Marketing	Steve	9,500
Marketing	Neha	10,500
Tech	Cristina	9,000
Tech	Kari	10,000
Tech	Mohit	8,000

Find share of each employee in their department's payout

Department	Employee	Salary	Dept Payout
Sales	Anna	8,000	18,000
Sales	Ivan	10,000	18,000
Marketing	Steve	9,500	20,000
Marketing	Neha	10,500	20,000
Tech	Cristina	9,000	27,000
Tech	Kari	10,000	27,000
Tech	Mohit	8,000	27,000

Share % = Salary * 100 / DeptPayout

Find share of each employee in their department's payout

Department	Employee	Salary	Dept Payout	Share %
Sales	Anna	8,000	18,000	44.4
Sales	Ivan	10,000	18,000	55.6
Marketing	Steve	9,500	20,000	47.5
Marketing	Neha	10,500	20,000	52.5
Tech	Cristina	9,000	27,000	33.3
Tech	Kari	10,000	27,000	37.0
Tech	Mohit	8,000	27,000	26.7

Share % = Salary * 100 / DeptPayout

Find share of each employee in their department's payout

Department	Employee	Salary	Dept Payout	Share %
Sales	Anna	8,000	18,000	44.4
Sales	Ivan	10,000	18,000	55.6
Marketing	Steve	9,500	20,000	47.5
Marketing	Neha	10,500	20,000	52.5

```
SELECT *
        , Salary * 100 / DeptPayout                                    AS SharePercent
FROM (
            SELECT Department, Employee, Salary
                    , SUM (Salary)      OVER ( PARTITION BY Department )   AS DeptPayout
            FROM Departments
        ) subquery
```

Summary

Create temp views on DataFrames & run SQL queries

Spark provides built-in catalogs (local & Hive)

- Data for Spark tables is stored in Storage as files
- Only metadata of dataset is stored in catalog

Write complex logic as UDF & invoke with DF/SQL code

Operations on multiple datasets

- Join operations – Inner, Outer & Cross joins
- Set operations – Union (All), Intersect, Except

Use Windows to aggregate on subset of related rows

- Window can be divided into partitions
- Use Over & PartitionBy clauses to define window/partitions

Up Next:
Performing Optimizations in Spark