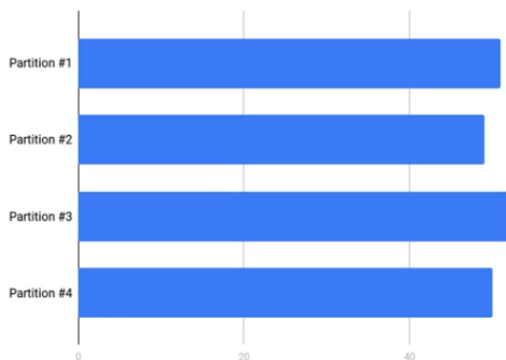


SKEW IN SPARK PROGRAMMING

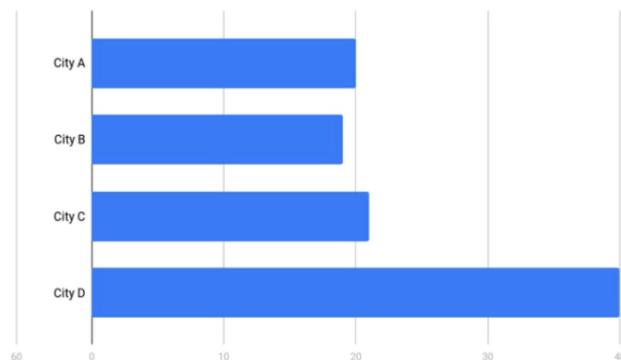
What is Skew

- ✚ “Data Skew” is a Condition in which a Table’s data is “Unevenly Distributed” among the Partitions in the Cluster. Operations, such as “Join” performs very slow on such Partitions.
- ✚ Initially, data is typically read in “Partitions” of 128 MB size each, and, evenly distributed throughout the “Cluster of Computers”.
- ✚ However, as the data is “Transformed”, for example “Aggregated”, it is possible to have significantly more records in one “Partition” than another. This is called “Skew”. To some degrees, a small amount of “Skew” is ignorable, may be in 10% range. But large “Skews” can result in “Spill”, or even worse, really hard to diagnose “Out of Memory Errors”.

Before aggregation



After aggregation by city



- ✚ As seen from the above image, after “Aggregated” operation, if the data in City D is 2 times larger than the data in City A, City B, or City C -
 - The data in City D is going to take 2 times as long to process than the data in City A, City B, or City C, assuming it takes “n” seconds per Record.
 - The data in City D requires twice as much RAM to process than the data in City A, City B, or City C, assuming it takes “n” MB per Record.

The ramification of “Skew” -

- The entire “Stage” is going to take as long as the “Longest-Running Task”.
 - Purely from an Execution Standpoint in Time, if all the four Partitions are processed simultaneously, the entire Job will take as long as the processing of the slowest Stage, i.e., the Stage with the Task of processing of City D.
- The more dire problem would be to not have enough RAM for the “Skewed Partitions”.

- If the Cluster is not created of proper size with keeping “Skewed Partitions” in mind, then it might happen that the Maximum Capacity of the Executors exceeds, in terms of Handling the RAM, required to Process Data in that Skewed Partition.

✚ **Skew - Speed vs. RAM** - When the “Skew” in a “Dataset” is significant enough to cause a “Performance Problem”, it generally manifests itself in one of the following two ways -

- **Speed** - Because of the “Skew” in a Dataset, a particular set of Tasks, or one Task out of a larger set of Tasks may take longer than it is expected to process.
- **RAM** - Because of the “Skew” in a Dataset, there might be a Problem with RAM. The RAM problem manifests itself as “Spill”, or, in worst case scenario, “Out of Memory (OOM) Errors”.

In such cases, where “Skew” in a “Dataset” causes a “Performance Problem”, the first problem to solve is the “Uneven Distribution of records” across all “Partitions”, because, solving for either “Speed”, or, “RAM” will only treat the Symptoms, and not the Root Cause. “Uneven Distribution of records” manifests itself as proportionally “Slower Tasks”.


✚ **How to Find Skews in Query** - “Data Skew” can severely downgrade Performance of Queries, especially those with “Joins”. “Joins” between big Tables require “Shuffling Data” and the “Data Skew” can lead to an extreme imbalance of work in the Cluster. It’s like that “Data Skew” is affecting a Query, if a Query appears to be stuck finishing very few Tasks (e.g., the last 3 Tasks out of 200). To verify that “Data Skew” is affecting a Query -

- Click the Stage that is stuck and verify that it is doing a Join.
- After the Query finishes, find the Stage that does a Join, and check the “Task Duration Distribution”.
- Sort the Tasks by “Decreasing Duration” and check the first few Tasks. If one Task took much longer to complete than the other Tasks, there is “Data Skew”.

✚ **Different Ways to Mitigate Skews in Dataset** - Following are different “Strategies” for fixing “Skews” -

- Employ a Databricks-specific “Skew Hint”, called “Skew Join Optimization”.
- Enable “Adaptive Query Execution” in “Spark 3.0”.
- Salt the “Skewed Column” with a random number, which creates better “Distribution” across each “Partition” at the cost of extra processing.

What is Skew Join Optimization

 **How Data is Skewed** - The following code reads the Delta File from the provided Path. Then Groups the data by the Column “city_id”, performs a Count operation and lastly, performs a Sort operation upon the Grouped data.

`sc.setJobDescription("How Skewed?")`

```
dfVisualize = spark.read.\
    format("delta").\
    load(transactionPath).\
    groupBy("city_id").\
    count().\
    orderBy("count")
```

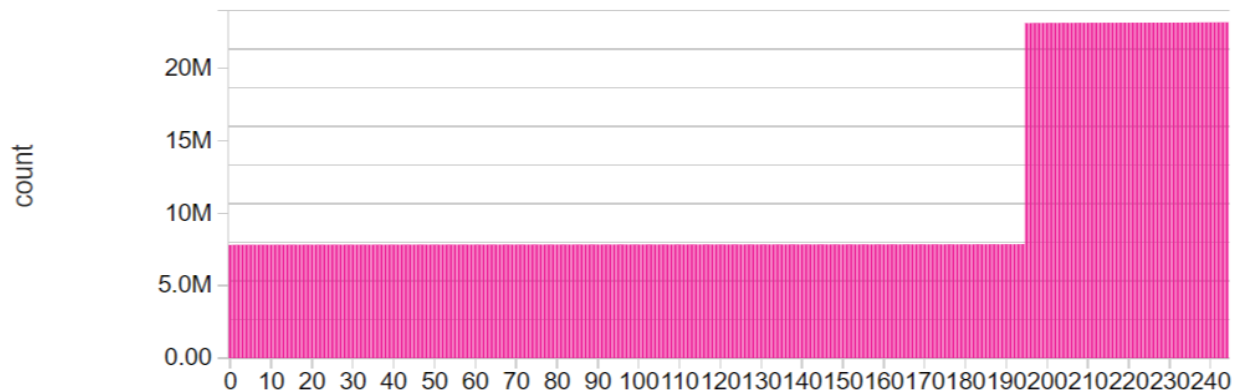
`display(dfVisualize)`

Display How "Data Skew" Occurs

```
1  sc.setJobDescription("How Skewed?")
2
3  dfVisualize = spark.read.\
4      format("delta").\
5      load(transactionPath).\
6      groupBy("city_id").\
7      count().\
8      orderBy("count")
9
10 display(dfVisualize)
```

The resultant data, in the created DataFrame, can be visually displayed in the following way -

```
dfVisualize: pyspark.sql.dataframe.DataFrame = [city_id: integer, count: long]
```



It can be seen that vast majority of Dataset is coming in around 8 million, and, a Subset of Dataset is coming in around 23 million. It means that there is a significant number of “Skewed Records”.

✚ **Create a Baseline** - Let's create the Baseline. Load two DataFrames. Perform a Join operation between the two DataFrames, based on the Column “city_id”. Then, perform a “noop” Write.

```
sc.setJobDescription("Establish a Baseline")
```

```
#Load the City DataFrame
```

```
dfCity = spark.read.\  
    format("delta").\  
    load(cityPath)
```

```
#Load the Transaction DataFrame
```

```
dfTransaction = spark.read.\  
    format("delta").\  
    load(transactionPath)
```

```
#Join the Two DataFrames by "city_id" and Execute a "noop" Write
```

```
dfTransaction.\  
    join(dfCity, dfTransaction["city_id"] == dfCity["city_id"]).\  
    write.\  
    format("noop").\  
    mode("overwrite").\  
    save()
```

Perform Join - Baseline

```
1  sc.setJobDescription("Establish a Baseline")
2
3  #Load the City DataFrame
4  dfCity = spark.read.\
5          format("delta").\
6          load(cityPath)
7
8  #Load the Transaction DataFrame
9  dfTransaction = spark.read.\
10         format("delta").\
11         load(transactionPath)
12
13 #Join the Two DataFrames by "city_id" and Execute a "noop" Write
14 dfTransaction.\
15     join(dfCity, dfTransaction["city_id"] == dfCity["city_id"]).\
16     write.\
17     format("noop").\
18     mode("overwrite").\
19     save()
```

Command took 7.99 minutes -- by chakraborty.oindrila [REDACTED] at 1/6/2022, 1:27:09 PM on [REDACTED]

From the above image, it could be seen that it took almost “8 minutes” to process the Join operation between the two DataFrames without “Skew Hint”.

✚ **Specify Skew Hints in Dataset and DataFrame-Based Join Commands** - When a Join command is performed with DataFrame, or, Dataset Objects, if it is found that the Query is stuck on finishing a small number of Tasks due to “Data Skew”, the “Skew Hint” can be specified using the “*hint (“skew”)*” method, i.e., “*df.hint (“skew”)*”. The “Skew Join Optimization” is performed on the DataFrame for which the “Skew Hint” is specified.

With the information from a “Skew Hint”, Databricks Runtime can construct a better Query Plan, one that does not suffer from “Data Skew”.

In addition to the basic “Hint”, it is possible to specify the “*hint ()*” method with the following combinations of Parameters -

- **DataFrame With Single Column Name** - The “Skew Join Optimization” is performed on the specified Column of the DataFrame.
df.hint (“skew”, “col1”)

- DataFrame With Multiple Column Names - The “Skew Join Optimization” is performed for multiple specified Columns of the DataFrame.

`df.hint("skew", ["col1", "col2"])`

- DataFrame With Single Column Name and Skew Value - The “Skew Join Optimization” is performed on the data in the specified Column of the DataFrame with the “Skew Value”.

`df.hint("skew", "col1", "value")`

It is possible to specify the “Skew Hint” for multiple DataFrame Objects, involved in a Join operation, in the following way -

`df1.hint("skew", "col1").join(df2.hint("skew", "col2"), df1["col1"] == df2["col2"])`

Let’s create the second DataFrame with “Skew Hint”, because we found that “Data Skew” is present in the second DataFrame. Load two DataFrames. Perform a Join operation between the two DataFrames, based on the Column “city_id”. Then, perform a “noop” Write.

`sc.setJobDescription("Join with Skew Hint")`

#Load the City DataFrame

```
dfCity = spark.read.\
    format("delta").\
    load(cityPath)
```

#Load the Transaction DataFrame

```
dfTransaction = spark.read.\
    format("delta").\
    load(transactionPath).\
    hint("skew", "city_id")
```

#Join the Two DataFrames by "city_id" and Execute a "noop" Write

```
dfTransaction.\
    join(dfCity, dfTransaction["city_id"] == dfCity["city_id"]).\
    write.\
    format("noop").\
    mode("overwrite").\
    save()
```

Perform Join - Skew Hint

```
1  sc.setJobDescription("Join with Skew Hint")
2
3  #Load the City DataFrame
4  dfCity = spark.read.\
5          format("delta").\
6          load(cityPath)
7
8  #Load the Transaction DataFrame
9  dfTransaction = spark.read.\
10         format("delta").\
11         load(transactionPath).\
12         hint("skew", "city_id")
13
14 #Join the Two DataFrames by "city_id" and Execute a "noop" Write
15 dfTransaction.\
16     join(dfCity, dfTransaction["city_id"] == dfCity["city_id"]).\
17     write.\
18     format("noop").\
19     mode("overwrite").\
20     save()
```

Command took 6.72 minutes -- by chakraborty.oindrila at 1/6/2022, 1:27:09 PM on

From the above image, it could be seen that it took almost “7 minutes” to process the Join operation between the two DataFrames with “Skew Hint”.

Following is the comparison of Metrics for the code “Join - Baseline” with the code “Join - Skew Hint” -

Code Type	Duration	Tasks	Health	Shuffle	Spill
Standard	~8 min	832	Bad	0 / 0 / ~100 KB / ~400 MB / ~3 GB	~50 GB
Skew Hint	~7 min	832	Mostly OK	134 MB / 174 MB / 184 MB / 195 MB / 1.1 GB	~4 GB

- **Special Note** - Sometimes, even though there is less Spill when “Skew Hint” is used on DataFrame or Dataset, it is slower. This might be because of the acts of “Re-Partitioning the Data” and “Re-Organizing the Data” that are done under the hood by Apache Spark. So, using “Skew Hint” on a DataFrame introduces Performance Hit.

Configure Skew Hint for Join with Relation (Table / View / Subquery) in Spark SQL -

- Configure Skew Hint with Only Relation Name - A “Skew Hint” must contain at least the name of the “Relation” with “Skew”. A “Relation” is a “Table”, “View”, or a “Subquery”. All Joins with the “Relation” then use “Skew Join Optimization”.

- ✓ Table with Skew - Spark SQL Query of Join between two Tables/Views without Skew Hint” -

%sql

```
SELECT          A.city_id, B.city, B.country, A.transacted_at,
A.trx_id, A.retailer_id, A.description, A.amount
FROM            global_temp.tblTransaction A
INNER JOIN      global_temp.tblCity B
ON              A.city_id = B.city_id
```

Perform Join between Tables/Views - Baseline

```
1 %sql
2 SELECT      A.city_id, B.city, B.country, A.transacted_at, A.trx_id, A.retailer_id, A.description, A.amount
3 FROM        global_temp.tblTransaction A
4 INNER JOIN  global_temp.tblCity B
5 ON          A.city_id = B.city_id
```

Command took 1.95 minutes -- by chakraborty.oindrila [REDACTED] at 1/6/2022, 1:54:20 PM on [REDACTED]

From the above image, it could be seen that it took almost “2 minutes” to process the Join operation between the two Tables/Views without “Skew Hint”.

Spark SQL Query of Join between two Tables/Views with Skew Hint” -

%sql

```
SELECT          /*+  SKEW('A')  */ A.city_id, B.city,
B.country, A.transacted_at, A.trx_id, A.retailer_id, A.description,
A.amount
FROM            global_temp.tblTransaction A
INNER JOIN      global_temp.tblCity B
ON              A.city_id = B.city_id
```

Perform Join between Tables/Views - Skew Hint

SQL ▶ ▼ - ✕

```
1 %sql
2 SELECT      /*+  SKEW('A')  */ A.city_id, B.city, B.country, A.transacted_at, A.trx_id, A.retailer_id, A.description, A.amount
3 FROM        global_temp.tblTransaction A
4 INNER JOIN  global_temp.tblCity B
5 ON          A.city_id = B.city_id
```

Command took 4.93 minutes -- by chakraborty.oindrila [REDACTED] at 1/6/2022, 2:01:48 PM on [REDACTED]

From the above image, it could be seen that it took almost “5 minutes” to process the Join operation between the two Tables/Views with “Skew Hint”.

In this case, using “Skew Hint” on a Relation (Table/View/Subquery) improves Data Skew but introduces Performance Hit.

- ✓ **Subquery with Skew** - Spark SQL Query of Join between a Table/View and a Subquery without Skew Hint” -

%sql

```
SELECT                A.city_id, B.city, B.country, A.transacted_at,
A.trx_id, A.retailer_id, A.description, A.amount
FROM
( SELECT                *
  FROM                tblTransaction
) A
INNER JOIN            tblCity B
ON                    A.city_id = B.city_id
```

Perform Join between Tables/Views and Subquery - Baseline

```
1 %sql
2 SELECT                A.city_id, B.city, B.country, A.transacted_at, A.trx_id, A.retailer_id, A.description, A.amount
3 FROM
4 ( SELECT                *
5   FROM                tblTransaction
6 ) A
7 INNER JOIN            tblCity B
8 ON                    A.city_id = B.city_id
```

Command took 8.13 minutes -- by chakraborty.oindrila [REDACTED] at 1/6/2022, 8:12:04 PM on [REDACTED]

From the above image, it could be seen that it took “8.13 minutes” to process the Join operation between a Table/View and a Subquery without “Skew Hint”.

Spark SQL Query of Join between a Table/View and a Subquery with Skew Hint” -

%sql

```
SELECT                /*+ SKEW('A') */ A.city_id, B.city,
B.country, A.transacted_at, A.trx_id, A.retailer_id, A.description,
A.amount
FROM
( SELECT                *
```

```

FROM                                tblTransaction
) A
INNER JOIN                          tblCity B
ON                                  A.city_id = B.city_id

```

Perform Join between Tables/Views and Subquery - Skew Hint

```

1 %sql
2 SELECT      /*+ SKEW('A') */ A.city_id, B.city, B.country, A.transacted_at, A.trx_id, A.retailer_id, A.description, A.amount
3 FROM
4 (   SELECT      *
5     FROM        tblTransaction
6 ) A
7 INNER JOIN      tblCity B
8 ON              A.city_id = B.city_id

```

Command took 8.68 minutes -- by chakraborty.oindrila [REDACTED] at 1/6/2022, 8:32:51 PM on [REDACTED]

From the above image, it could be seen that it took “8.68 minutes” to process the Join operation between a Table/View and a Subquery with “Skew Hint”.

In this case, using “Skew Hint” on a Relation (Table/View/Subquery) improves Data Skew but introduces Performance Hit.

- **Configure Skew Hint with Relation Name and Column Names** - There might be multiple Joins on a “Relation” and only some of the Joins will suffer from Skew. “Skew Join Optimization” has some overhead. So, it is better to use it only when needed. For this purpose, the “Skew Hint” accepts Column Names. Only the Joins with the provided Columns use “Skew Join Optimization”.

- ✓ **Skew on Single Column** - Spark SQL Query of Join on a Single Column between two Tables/Views with Skew Hint” -

```

%sql
SELECT      /*+ SKEW('A', 'city_id') */ A.city_id, B.city,
B.country, A.transacted_at, A.trx_id, A.retailer_id, A.description,
A.amount
FROM        global_temp.tblTransaction A
INNER JOIN   global_temp.tblCity B
ON          A.city_id = B.city_id

```

Perform Join on a Single Column between Tables/Views - Skew Hint

```

1 %sql
2 SELECT      /*+ SKEW('A', 'city_id') */ A.city_id, B.city, B.country, A.transacted_at, A.trx_id, A.retailer_id,
A.description, A.amount
3 FROM        global_temp.tblTransaction A
4 INNER JOIN   global_temp.tblCity B
5 ON          A.city_id = B.city_id

```

- ✓ **Skew on Multiple Columns** - Spark SQL Query of Join on Multiple Columns between two Tables/Views with Skew Hint” -

```
%sql
SELECT                               /*+ SKEW('A', ('city_id', 'city')) */ A.city_id,
B.city, B.country, A.transacted_at, A.trx_id, A.retailer_id, A.description,
A.amount
FROM                                global_temp.tblTransaction A
INNER JOIN                          global_temp.tblCity B
ON                                  A.city_id = B.city_id
```

Perform Join on Multiple Columns between Tables/Views - Skew Hint

```
1 %sql
2 SELECT                               /*+ SKEW('A', ('city_id', 'city')) */ A.city_id, B.city, B.country, A.transacted_at, A.trx_id, A.retailer_id,
  A.description, A.amount
3 FROM                                global_temp.tblTransaction A
4 INNER JOIN                          global_temp.tblCity B
5 ON                                  A.city_id = B.city_id
```

- **Configure Skew Hint with Relation Name, Column Names and Skew Values** - It is possible to specify Skew Values in the “Skew Hint”. Depending on the Query and the Data, the Skew Values might be known or might be easy to find out, as the Skew Values never change. Doing this reduces the overhead of “Skew Join Optimization”. Otherwise, “Delta Lake” detects the Skew Values automatically.

- ✓ **Skew on Single Column and Single Skew Value** - Spark SQL Query of Join on a Single Column and a Single Skew Value between two Tables/Views with Skew Hint” -

```
%sql
SELECT                               /*+ SKEW('A', 'city_id', 28424447) */
A.city_id, B.city, B.country, A.transacted_at, A.trx_id, A.retailer_id,
A.description, A.amount
FROM                                global_temp.tblTransaction A
INNER JOIN                          global_temp.tblCity B
ON                                  A.city_id = B.city_id
```

Perform Join on a Single Column with Single Skew Value between Tables/Views -

```
1 %sql
2 SELECT                               /*+ SKEW('A', 'city_id', 28424447) */ A.city_id, B.city, B.country, A.transacted_at, A.trx_id, A.retailer_id,
  A.description, A.amount
3 FROM                                global_temp.tblTransaction A
4 INNER JOIN                          global_temp.tblCity B
5 ON                                  A.city_id = B.city_id
```

- ✓ **Skew on Single Column and Multiple Skew Values** - Spark SQL Query of Join on a Single Column and Multiple Skew Values between two Tables/Views with Skew Hint” -

%sql

```
SELECT /*+ SKEW('A', 'city_id', (28424447, 559832710)) */ A.city_id, B.city, B.country, A.transacted_at, A.trx_id, A.retailer_id, A.description, A.amount
FROM global_temp.tblTransaction A
INNER JOIN global_temp.tblCity B
ON A.city_id = B.city_id
```

Perform Join on a Single Column with Multiple Skew Values between Tables/View

```
1 %sql
2 SELECT /*+ SKEW('A', 'city_id', (28424447, 559832710)) */ A.city_id, B.city, B.country, A.transacted_at, A.trx_id,
3 A.retailer_id, A.description, A.amount
4 FROM global_temp.tblTransaction A
5 INNER JOIN global_temp.tblCity B
ON A.city_id = B.city_id
```

- ✓ **Skew on Multiple Columns and Multiple Skew Values** - Spark SQL Query of Join on Multiple Columns and Multiple Skew Values between two Tables/Views with Skew Hint” -

%sql

```
SELECT /*+ SKEW('A', ('city_id', 'city'), ((28424447, 'Albany'), (559832710, 'Jackson')) */ A.city_id, B.city, B.country, A.transacted_at, A.trx_id, A.retailer_id, A.description, A.amount
FROM global_temp.tblTransaction A
INNER JOIN global_temp.tblCity B
ON A.city_id = B.city_id
```

Perform Join on Multiple Columns with Multiple Skew Values between Tables/View

```
1 %sql
2 SELECT /*+ SKEW('A', ('city_id', 'city'), ((28424447, 'Albany'), (559832710, 'Jackson')) */ A.city_id, B.city,
3 B.country, A.transacted_at, A.trx_id, A.retailer_id, A.description, A.amount
4 FROM global_temp.tblTransaction A
5 INNER JOIN global_temp.tblCity B
ON A.city_id = B.city_id
```

Enabling Adaptive Query Execution for Skew Join

- ✚ It is recommended to rely on Adaptive Query Execution (AQE) Skew Join Handling, rather than using the Skew Join Hint, because AQE Skew Join is completely Automatic, and in general, performs better than the Skew Hint counterpart.

✚ By enabling the Adaptive Query Execution (AQE), Apache Spark checks the “Stage Statistics” and determines if there are any “Skew Joins”. If found, Apache Spark Optimizes the “Skew Joins” by Splitting the Bigger Partitions into Smaller Evenly Sized Partitions. Then Spark performs Join of these Smaller Partitions with the corresponding Matching Partition Size of other Table/DataFrame.

✚ There are two Size Conditions that must be satisfied for AQE to detect a Partition as a Skewed Partition -

- The Skewed Partition Size should be Larger Than the “[spark.sql.adaptive.skewJoin.skewedPartitionThresholdInBytes](#)”. The default value is “**256 MB**”.

As Apache Spark Splits the Skewed Partitions by Targeting the Value of the Config Property “[spark.sql.adaptive.advisoryPartitionSizeInBytes](#)”, ideally the Value of the Config Property “[skewedPartitionThresholdInBytes](#)” should be Larger Than the Value of the Config Property “[advisoryPartitionSizeInBytes](#)”. So, anytime the Value of the Config Property “[advisoryPartitionSizeInBytes](#)” is increased, the Value of the Config Property “[skewedPartitionThresholdInBytes](#)” should also be increased.

- The Skewed Partition Size should be Larger Than the -

✓ [Median Size of all Partitions * Skewed Partition Factor](#)

“Skewed Partition Factor” means the Configuration Property “[spark.sql.adaptive.skewJoin.skewedPartitionFactor](#)”. The default value is “**5**”.

✚ To enable the AQE for Optimizing the “Skew Joins”, both of the following Configuration Properties need to be set to “True” -

- [spark.conf.set\("spark.sql.adaptive.enabled", True\)](#)
- [spark.conf.set\("spark.sql.adaptive.skewJoin.enabled", True\)](#)

✚ There is another Configuration Property that goes with the above mentioned AQE Configuration Properties, i.e., “[spark.sql.adaptive.advisoryPartitionSizeInBytes](#)”. When AQE is enabled, as Adaptive Query Engine performs Re-Partitioning on the Data, Apache Spark makes sure to Avoid creating too many Small Tasks using this Configuration Property. Spark will Coalesce “Contiguous Shuffle Partitions” according to the Target Size specified by “[spark.sql.adaptive.advisoryPartitionSizeInBytes](#)”. The default value is “**64 MB**”.

[#Enable AQE and the Adaptive Skew Join](#)

[spark.conf.set\("spark.sql.adaptive.enabled", True\)](#)

[spark.conf.set\("spark.sql.adaptive.skewJoin.enabled", True\)](#)

[#Default is 64 MB. In this case, maintain 128 MB as the Least Size of Coalesce Shuffle Partition](#)

[spark.conf.set\("spark.sql.adaptive.advisoryPartitionSizeInBytes", "128m"\)](#)

Enable AQE to Optimize Skew Join

```
1  sc.setJobDescription("Join with AQE")
2
3  #Enable AQE and the Adaptive Skew Join
4  spark.conf.set("spark.sql.adaptive.enabled", True)
5  spark.conf.set("spark.sql.adaptive.skewJoin.enabled", True)
6
7  #Default is 64 MB. In this case, maintain 128 MB as the Least Size of Coalesce Shuffle Partition
8  spark.conf.set("spark.sql.adaptive.advisoryPartitionSizeInBytes", "128m")
```

- ✚ In the “Join”, using the DataFrame, no “Hint” should be provided -
`sc.setJobDescription("Join with AQE")`

#Load the City DataFrame

```
dfCity = spark.read.\
    format("delta").\
    load(cityPath)
```

#Load the Transaction DataFrame

```
dfTransaction = spark.read.\
    format("delta").\
    load(transactionPath)
```

*#Join the Two DataFrames by "city_id" and Execute a "noop" Write
dfTransaction.*

```
    join(dfCity, dfTransaction["city_id"] == dfCity["city_id"]).\
    write.\
    format("noop").\
    mode("overwrite").\
    save()
```

Perform Join - AQE

```
1  sc.setJobDescription("Join with AQE")
2
3  #Load the City DataFrame
4  dfCity = spark.read.\
5      format("delta").\
6      load(cityPath)
7
8  #Load the Transaction DataFrame
9  dfTransaction = spark.read.\
10     format("delta").\
11     load(transactionPath)
12
13 #Join the Two DataFrames by "city_id" and Execute a "noop" Write
14 dfTransaction.\
15     join(dfCity, dfTransaction["city_id"] == dfCity["city_id"]).\
16     write.\
17     format("noop").\
18     mode("overwrite").\
19     save()
```

Code Type	Duration	Tasks	Health	Shuffle	Spill
Standard	~8 min	832	Bad	0 / 0 / ~100 KB / ~400 MB / ~3 GB	~50 GB
Skew Hint	~7 min	832	Mostly OK	134 MB / 174 MB / 184 MB / 195 MB / 1.1 GB	~4 GB
AQE	~6 min	1489	Excellent	0 / ~115 MB / ~115 KB / ~125 MB / ~130 MB	0

✚ **Limitation of AQE for Skew Join** - Using AQE, the Skew Join Handling support is Limited to only certain Join Types. Example - in “Left Outer Join”, the Skew on the Left Side can only be Optimized.

Salted Join for Skew

- ✚ Prior to Spark 3.0, i.e., when the Adaptive Query Execution (AQE) was not available to be Enabled, and, if the “Skew Hints” are not available in the Databricks, for some reason, then the “Skew-Salted Join” is the only option to solve the Skew Join problem.
- ✚ This approach is by far the Most Complicated to implement. This scenario is mostly implemented by Data Engineers.

- ✚ This approach can take Longer to Execute in some cases, because the act of Salting involves lots of Re-Partitioning.
- ✚ This approach remains a Viable Option when other two Solutions are Not Available.
- ✚ The Idea behind this approach is to Split the Large Partitions into Smaller Ones using a “Salt”, i.e., a Random Value is added to each of the Partitioning Columns so that all the Matching Data naturally falls into the Same Partition.
- ✚ This approach is more about Guaranteeing the Execution of all Tasks, and Not a Uniform Duration of Each Task.
- ✚ **Step 1** - Create a DataFrame Based on the Range of the “Skew Factor”. Here “7” is used as the “Skew Factor”. The resultant DataFrame will have 7 Values, 0 to 6.
`sc.setJobDescription("Create Salt DataFrame")`

```
spark.conf.set("spark.sql.adaptive.enabled", False)
spark.conf.set("spark.sql.adaptive.skewJoin.enabled", False)
```

```
# Too Large - "Unnecessary Overhead" in the "Join" and "Cross Join"
# Too Small - "Skewed Partition" is not split up enough
# The Value "7" as "Skew Factor" was "Selected" after much experimentation
skewFactor = 7
saltDf = spark.range(skewFactor).toDF("salt")
```

Create a DataFrame Based on the Range of the “Skew Factor”

```
1  sc.setJobDescription("Create Salt DataFrame")
2
3  spark.conf.set("spark.sql.adaptive.enabled", False)
4  spark.conf.set("spark.sql.adaptive.skewJoin.enabled", False)
5
6  # Too Large - "Unnecessary Overhead" in the "Join" and "Cross Join"
7  # Too Small - "Skewed Partition" is not split up enough
8  # The Value "7" as "Skew Factor" was "Selected" after much experimentation
9  skewFactor = 7
10 saltDf = spark.range(skewFactor).toDF("salt")
```

- ✚ **Step 2** - Cross Join the Salt DataFrame “saltDf” with the Dimension DataFrame to create the “Salted Dimension DataFrame”. Re-Partitioning can help Mitigate the Spills and Evenly Re-Distributes the new Dimension Table across all Partitions. In this case “Re-Partition” is used to reduce the Impact of “Cross Join”, because the “Cross Join” would be an “Expensive Operation” if the “Skew Factor” is high.


```
sc.setJobDescription("Create Salted Dimension DataFrame")
```

```
# Post "Cross Join", we will be at ~865 MB  
# "128 MB" is Spark's Safe Default Partition Size  
noOfPartition = math.ceil(865/128)
```

```
#Load the Salted City DataFrame
```

```
dfSaltedCity = spark.read.\  
    format("delta").\  
    load(cityPath).\  
    repartition(noOfPartition).\  
    crossJoin(saltDf).\  
    withColumn("salted_city_id", concat("city_id", lit("_"),  
saltDf["salt"])).\  
    drop("salt")
```

Create the Salted Dimension DataFrame

```
1  sc.setJobDescription("Create Salted Dimension DataFrame")  
2  
3  # Post "Cross Join", we will be at ~865 MB  
4  # "128 MB" is Spark's Safe Default Partition Size  
5  noOfPartition = math.ceil(865/128)  
6  
7  #Load the Salted City DataFrame  
8  dfSaltedCity = spark.read.\  
9      format("delta").\  
10     load(cityPath).\  
11     repartition(noOfPartition).\  
12     crossJoin(saltDf).\  
13     withColumn("salted_city_id", concat("city_id", lit("_"), saltDf["salt"])).\  
14     drop("salt")
```

✚ **Step 3** - For the Fact DataFrame, randomly assign a "Salt" to each record.
`sc.setJobDescription("Create Salted Fact DataFrame")`


```
#Load the Salted Transaction DataFrame
```

```
dfSaltedTransaction = spark.read.\  
    format("delta").\  
    load(transactionPath).\  
    withColumn("salt", (lit(skewFactor) *  
rand()).cast(IntegerType())).\  
    *
```

```
col("salt"))).\
    withColumn("salted_city_id", concat(col("city_id"), lit("_"),
    drop("salt"))
```

Create the Salted Fact DataFrame

```
1  sc.setJobDescription("Create Salted Fact DataFrame")
2
3  #Load the Salted Transaction DataFrame
4  dfSaltedTransaction = spark.read.\
5      format("delta").\
6      load(transactionPath).\
7      withColumn("salt", (lit(skewFactor) * rand()).cast(IntegerType()).\
8      withColumn("salted_city_id", concat(col("city_id"), lit("_"), col("salt"))).\
9      drop("salt")
```

 **Step 4** - Join the two DataFrames based on the “salted_city_id” Column.
sc.setJobDescription("Perform the Salted Skew Join")

*#Join the Two Salted DataFrames by "salted_city_id" and Execute a "noop" Write
dfSaltedTransaction.*

```
    join(dfSaltedCity,      dfSaltedTransaction["salted_city_id"]      ==
dfSaltedCity["salted_city_id"]).\
    write.\
    format("noop").\
    mode("overwrite").\
    save()
```

Salted Skew-Join

```
1  sc.setJobDescription("Perform the Salted Skew Join")
2
3  #Join the Two Salted DataFrames by "salted_city_id" and Execute a "noop" Write
4  dfSaltedTransaction.\
5      join(dfSaltedCity, dfSaltedTransaction["salted_city_id"] == dfSaltedCity["salted_city_id"]).\
6      write.\
7      format("noop").\
8      mode("overwrite").\
9      save()
```

Code Type	Duration	Tasks	Health	Shuffle	Spill
Standard	~8 min	832	Bad	0 / 0 / ~100 KB / ~400 MB / ~3 GB	~50 GB
Skew Hint	~7 min	832	Mostly OK	134 MB / 174 MB / 184 MB / 195 MB / 1.1 GB	~4 GB
AQE	~6 min	1489	Excellent	0 / ~115 MB / ~115 KB / ~125 MB / ~130 MB	0
Salted	~10 min	832	Better/Bad	~400 MB / ~75 MB / ~150 KB / ~300 MB / ~800 MB	0

✚ **Limitation of Salting for Skew Join** - Salting a “Skewed Dataset” has several problems.

- “Salting” has the Side Effect of Splitting Smaller the Partitions into even Smaller Ones.
This is where the “Adaptive Query Execution (AQE)” shines as it does not have this Side Effect.
- “Salting” requires “Extra Work”. It is advised to use “Salting” to only the “Skewed Columns” in the entire Dataset. Data Engineers, who are preparing the Data for the Consumers, know which Columns in the Dataset are “Skewed”. So, even if it takes more time, the Data Engineers need to filter the Dataset into two halves - “Skewed” half, and the “Non-Skewed” half. Then, perform “Salting” on the “Skewed” half. Once it is “Salted” properly, “Join” the “Salted Skewed” half back to the “Non-Skewed” half. In this way, the “Salting” provides better performance for the Consumers, but it is a lot of extra work.