# Introduction

## What is Spark SQL?

Spark SQL is used for structured data processing in Spark.

The biggest abstraction in the Spark SQL API is the DataFrame

Spark SQL supports JDBC

It also supports Different RDBMS and No SQL Databases.

#### What is Spark Session?

Note: Spark session is like Spark context of Spark core and Spark streaming context of spark streaming.

1. We can access all functionality of Spark SQL through a spark session.
2. Data frames are created through Spark Session
3. Spark session provides a standard interface to work with different datasources
4. Using spark session can register data frames as temp tables and then run SQL on them.
5. **What is SQLContext in spark SQL?**

It’s the entry point for working with structured data (rows and columns) in Apache Spark.

Using SQLContext we can create DataFrame objects as well as the execution of SQL queries.

1. **What is HiveContext in spark SQL?**

HiveContex is used towork with Hive tables, a subclass of SQLContext

## What is DataFrame?

Data frame is distributed collection of data organized as rows and columns Data frame will have schema- column names and data types

Data Frame is developed on top of RDD.

Data frame is equivalent to Relational table.

DataFrames in Spark have the same capabilities as RDDs, such as immutability, in memory, resilient, distributed computing

## 4. What is dataset?

Datasets is like data frame which extends the benefit of compile-time type safety. That means it can analyse the applications for errors before they run.

## 5. Difference between RDD, Data frame, Dataset? IMP

|  |  |  |  |
| --- | --- | --- | --- |
| Type | RDD | Data frame | Dataset |
| **Data representation** | RDD is distributed dataset but it will not have any schema | It is also the distributed collection organized into the named columns | It is an extension of Data frames with more features like type-safety and object-oriented interface. |
| **optimization** | No in-built optimization engine for RDDs. Developers need to write the optimized code themselves. | It uses a catalyst optimizer for optimization. | It also uses a catalyst optimizer for optimization purposes. |
| **Projection of Schema** | Here, we need to define the schema manually. | It will automatically find out the schema of the dataset. | It will also automatically find out the schema of the dataset by using the SQL Engine. |
| **Aggregation Operation** | RDD is slower than both Data frames and Datasets to perform simple operations like grouping the data. | It provides an easy API to perform aggregation operations. It performs aggregation faster than both RDDs and Datasets. | Dataset is faster than RDDs but a bit slower than Data frames. |

# Partitioning

## Range partition? IMP

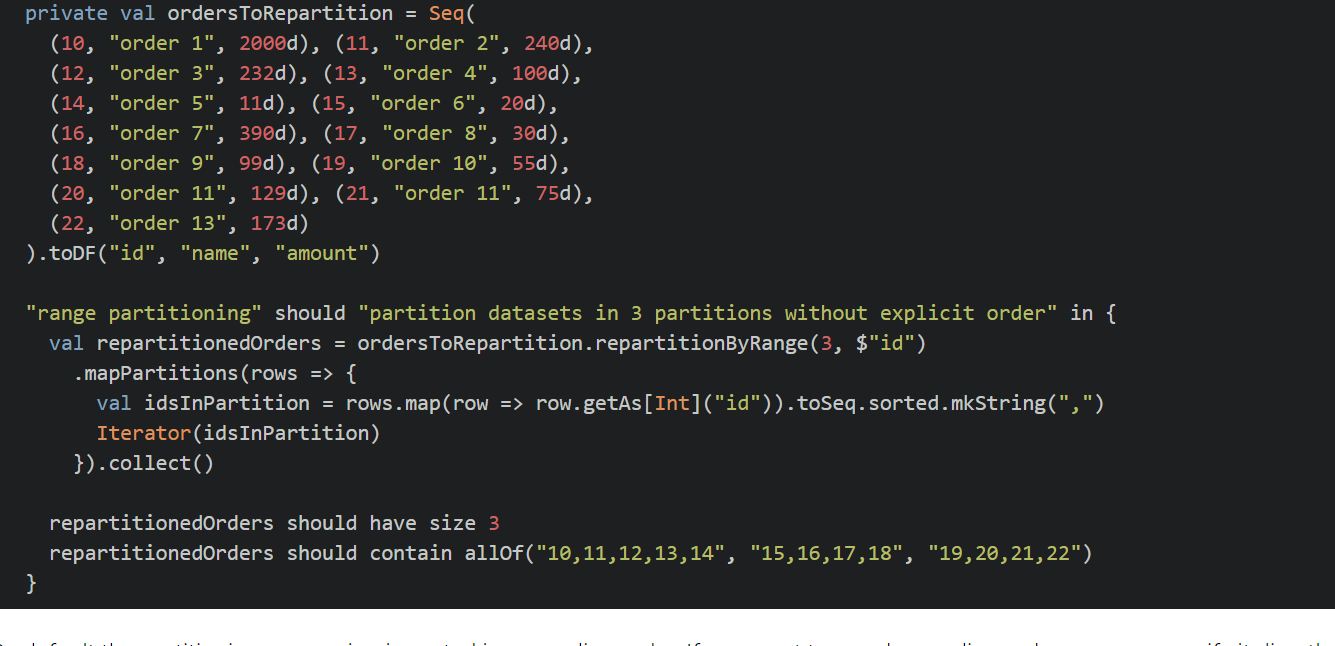
https://www.waitingforcode.com/apache-spark-sql/range-partitioning-apache-spark-sql/read#reparittionByRange

Range partition partitions data based on key range like age.

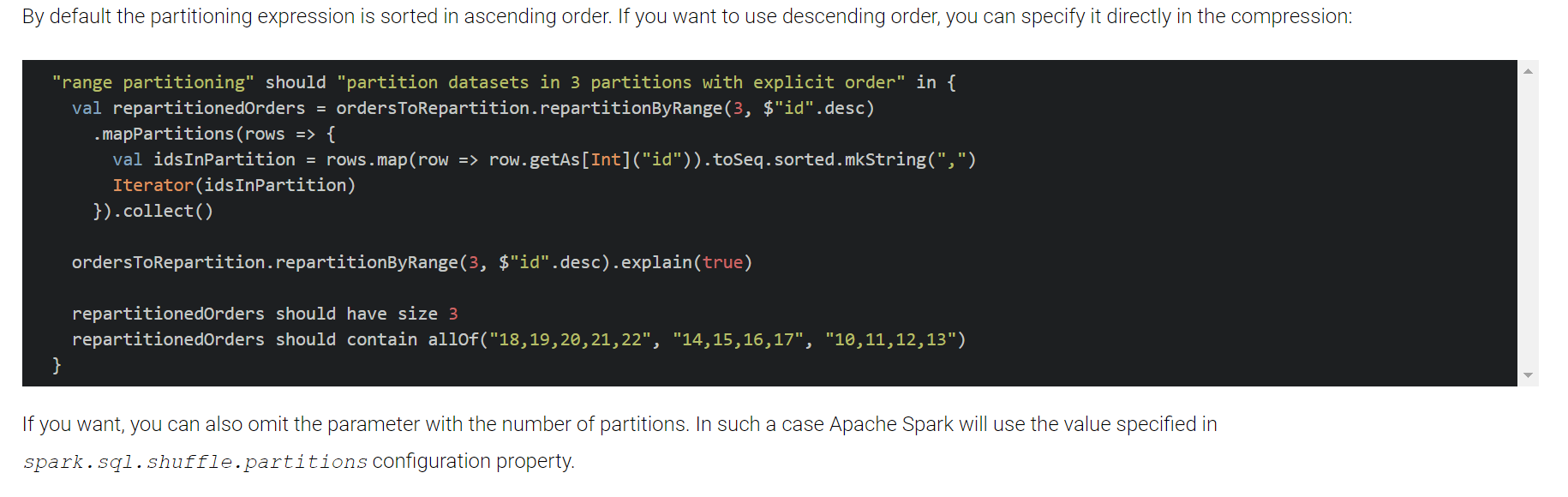
For example

0-20, 20-40

Apache Spark SQL implements range partitioning with repartitionByRange(numPartitions: Int, partitionExprs: Column\*) added in 2.3.0 version. When called, the function creates numPartitions of partitions based on the columns specified in partitionExprs, like in this snippet:



By default, the partitioning expression is sorted in ascending order. If you want to use descending order, you can specify it directly in the compression:

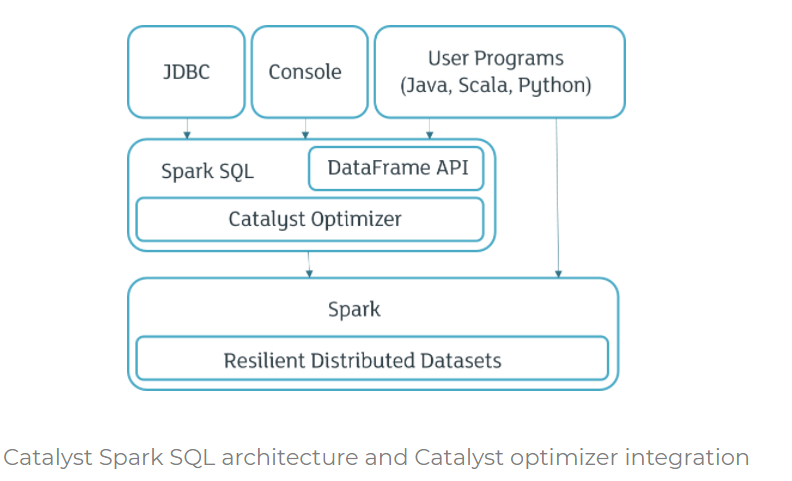


If you want, you can also omit the parameter with the number of partitions. In such a case Apache Spark will use the value specified in spark.sql.shuffle.partitions configuration property.

## 6. What is catalyst optimizer? IMP

*https://blog.bi-geek.com/en/spark-sql-optimizador-catalyst/*

Catalyst is a query plan optimizer. It helps to improve performance.



The Spark SQL Catalyst Optimizer improves developer productivity and the performance of their written queries. Catalyst automatically transforms relational queries to execute them more efficiently using techniques such as filtering, indexes and ensuring that data source joins are performed in the most efficient order

## 7. What are 2 main purpose of Catalyst optimizer? IMP

1. Add new optimization techniques to solve some problems with “big data”.

2. Apart from providing built in techniques It will allow developer to expand and customize the functions of the optimizer.

## 8. What are the components of catalyst optimizer? IMP

1. ***Trees***

The main data type in Catalyst is the tree. Each tree is composed of nodes, and each node has a node type and zero or more children These objects are immutable and can be manipulated with functional language.

As an example, let me show you the use of the following nodes:

Merge(Attribute(x), Merge(Literal(1), Literal(2))

Where:

Merge (left: Tree Node, right: Tree Node): mix of two expressions

Attribute (name: String): an attribute as input row

Literal (value: Int): a constant value

1. *Rules*

Trees can be manipulated using rules, which are functions of a tree to another tree. The transformation method applies the pattern matching function recursively on all nodes of the tree transforming each pattern to the result.

Below there’s an example of a rule applied to a tree.

tree.transform {

case Merge(Literal(c1), Literal(c2)) => Literal(c1) + Literal(c2)

}

It provides a general framework for transforming trees, which performs analysis/evaluation, optimization, planning, and runtime code spawning.

## 9. What are the types of optimization supported by catalyst optimizers?

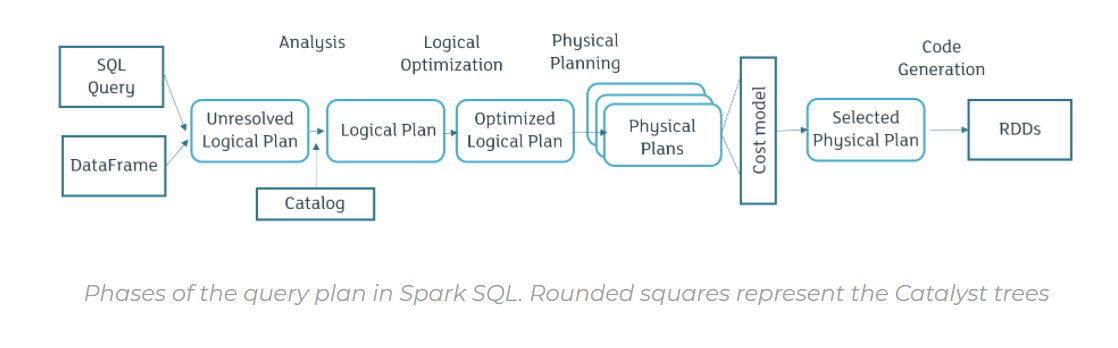
1. *Rule based optimization*

Indicated how to execute query from set of defined rules.

1. *Cost based optimization*

generates multiple execution plans and compares them to choose the lowest cost one

## 10. what are the 4 phases of transformation that catalyst perform?



1. ***Analysis***

Analysis will be done on AST (abstract syntax tree) returned by an SQL parser, and from a Data Frame object of the Spark SQL API. After analysis logical plan will be created.

1. ***Logic Optimization Plan***

In this phase, rule-based optimization is applied to the logical plan. It is possible to easily add new rules.

1. ***Physical plan***

In the physical plan phase, Spark SQL takes the logical plan and generates one or more physical plans using the physical operators that match the Spark execution engine. The plan to be executed is selected using the cost-based model (comparison between model costs).

1. ***Code generation***

To run on each machine, it is necessary to generate Java code bytecode.

## 11. How can you see logical and physical plan?

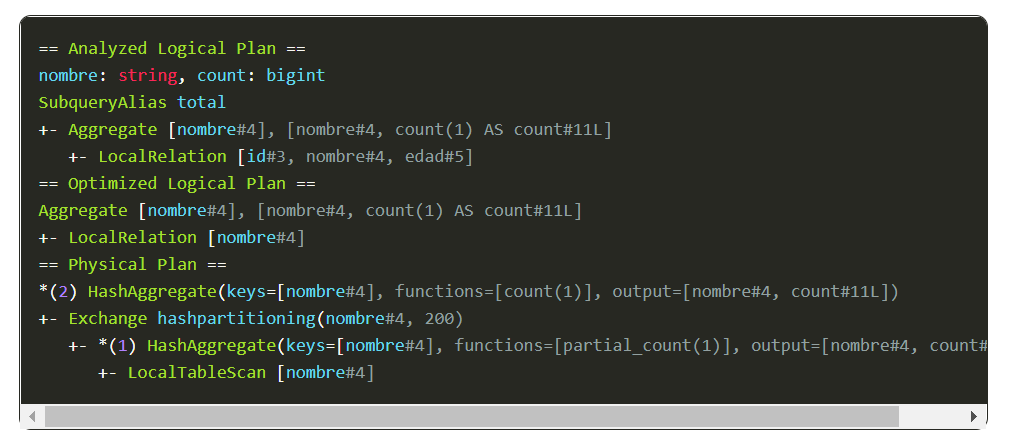
We can see logical and physical plan using **explain** method of DataSet

**Void explain ():** Prints the physical plan to the console for debugging purposes.

**void explain (Boolean extended):** Prints the plans (logical and physical) to the console for debugging purposes.



Below is the result



## 12. What is tungsten serializer in spark? IMP

We know that Spark workloads are increasingly bottlenecked by CPU and memory use rather than IO and network communication.

The goal of tungsten substantially improves the memory and CPU efficiency of the Spark applications and push the limits of the underlying hardware by changing apache spark execution engine.

Tungsten Project Includes These Initiatives:

* Memory Management and Binary Processing:

manage memory explicitly and eliminate the overhead of JVM object model and garbage collection

* Intermediate data in memory vs CPU registers:

Tungsten Phase 2 places intermediate data into CPU registers. This is an order of magnitudes reduction in the number of cycles to obtain data from the CPU registers instead of from memory

* Code generation

using code generation to exploit modern compilers and CPUs.

* Cache-aware computation

algorithms and data structures to exploit memory hierarchy

No virtual function dispatches: this reduces multiple CPU calls which can have a profound impact on performance when dispatching billions of times.

## How do you get right number of partitions?

Apache Spark can only run a single concurrent task for every partition of an RDD, up to the number of cores in your cluster (and probably 2-3x times that). Hence as far as choosing a “good” number of partitions, you generally want at least as many as the number of executors for parallelism. You can get this computed value by calling sc.defaultParallelism. The maximum size of a partition is ultimately limited by the available memory of an executor.

There are also cases where it’s not possible to which understand proper repartitioning key should be used for even data distribution. Hence, methods like Salting can be used which involves adding a new “fake” key and using alongside the current key for better distribution of data. Here’s an example:

Add a random element to large RDD and create new join key with it like “Salting key = actual join key + Random fake key where fake key takes value between 1 to N, with N being the level of distribution”

Add a random element to small RDD using a Cartesian product (1-N), to increase the number of entries and create new join key

Join RDDs on a new join key which will now be distributed better due to random seeding.

Remove the random fake key from the join key to get the final result of the join

In the example above, the fake key in the lookup dataset will be a Cartesian product (1-N), and for the main dataset, it will a random key (1-N) for the source data set on each row, and N being the level of distribution.

## Data Frame

1. **Ways to Create Data Frame**

Can create data frame or persist a data frame from variety of sources.

. CSV

. Data base Tables

. Hive/ NOSQL databases

. Json

. RDD

* **DataSet**
* **Catalyst optimizer**

1. **What is the difference between Aggregate, UDF and window function?**

*Aggregate*: calculates one result (sum/avg) for each group of rows.

*UDF*: calculates one result per row based on values of current row.

Window functions: calculates one result per row based on a window of rows.

# Performance

https://developer.ibm.com/technologies/artificial-intelligence/blogs/spark-performance-optimization-guidelines/

## What are the best practices for the performance tuning in spark? IMP

* **Parallelism or Partitioning**

We must make sure that our RDD is partitioned properly because partition is the parallelism factor. We must partition the data such that our cluster resource is utilized properly. If there are less partitions, then some executors may go idle. If there are too many partitions, then task scheduling will take time.

* **Tune cluster resources**

Tune the resources on the cluster depending on the resource manager and version of Spark.

Tune the available memory to the driver: spark.driver.memory.

Tune the number of executors and the memory and core usage based on resources in the cluster: executor-memory, num-executors, and executor-cores.

* **File Formats**

Apache parquet is best file format for spark as it gives very good read throughput, compression also spark works well with columnar formats and Apace Parquet is columnar. Spark also works well with ORC.

We must make sure there are not too many small files. If there are too many small files, we must do compaction for better performance.

* **Filter/Reduce dataset size:**

Look for opportunities to filter out data as early as possible in your application pipeline.

* **Lazy loading behaviour**

We know that in spark transformations are lazily evaluated and performed only when action is called so we need to make sure we will call any action when it is really needed.

* **Reduce shuffles:**
* **Avoid expensive operations**

Avoid order by if it is not needed.

When you are writing your queries, instead of using select \* to get all the columns, only retrieve the columns relevant for your query.

Don’t call count unnecessarily.

* **Cache appropriately**

Use caching when the same operation is computed multiple times in the pipeline flow.

* **Join**

Join is, in general, an expensive operation, so pay attention to the joins in your application to optimize them.

<https://spark.apache.org/docs/latest/configuration.html>

* **UDFs**

Spark has a number of built-in user-defined functions (UDFs) available. For performance, check to see if you can use one of the built-in functions since they are good for performance. Custom UDFs in the Scala API are more performant than Python UDFs. If you have to use the Python API, use the newly introduced pandas UDF in Python that was released in Spark 2.3. The pandas UDF (vectorized UDFs) support in Spark has significant performance improvements as opposed to writing a custom Python UDF. Get more information about writing a pandas UDF.

# Shuffling

https://umbertogriffo.gitbook.io/apache-spark-best-practices-and-tuning/rdd/avoiding\_shuffle\_less\_stage-\_more\_fast

## How will you avoid shuffling? IMP

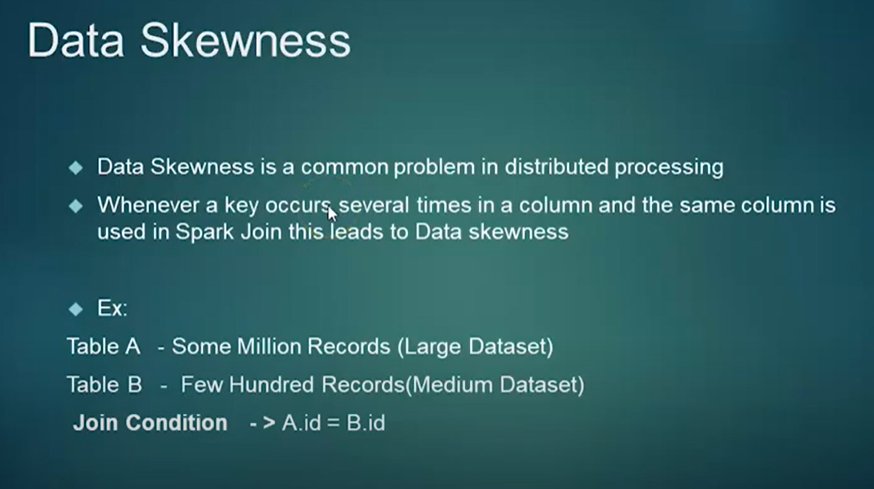
We know that all shuffle data must be written to disk and then transfer over network. If we avoid shuffling it will make our program run faster.

1. Better to have right partitioning when we read the data instead of repartitioning it later as it involves shuffling.
2. Partition the data such that each partition does not have big file sizes.
3. If some filtering operation is involved, then better to filter early than filtering data later.
4. We need to check if we can avoid operations like **repartition**, **join**, **cogroup** and any of the **By\*** or **ByKey\*** transformations can result in shuffling.
5. We should always avoid using out own aggregators we should always use our built in **aggrgateByKey**() operators.

# Data Skew

## What is data skew? IMP

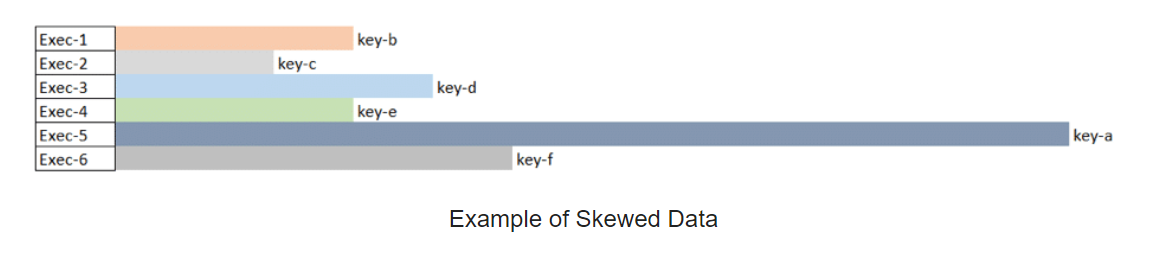
Uneven distribution of data across the partitions in a distributed environment is called skew.



## What problem does data skew can create? IMP

https://www.youtube.com/watch?v=d41\_X78ojCg

* Performance of spark job is impacted
* Resource allocated will be underutilized.
* We will lose advantage of distributed systems.
* Shuffling will be a problem



Here we see “key-a” has a larger amount of data in the partition so tasks on Exec-5 will take much longer to complete than the other five tasks.

Another important thing to remember is that Spark shuffle blocks can be no greater than 2 GB (internally because the ByteBuffer abstraction has a MAX\_SIZE set to 2GB). For example, if you are running an operation such as aggregations, joins or cache operations, a Spark shuffle will occur and having a small number of partitions or data skews can cause a high shuffle block issue. Hence, if you started seeing an error related to breach of MAX\_SIZE limits due to shuffle you know why it’s happening as it may be tied to skewed data.

## How will you identify in spark that data skewness is happening?

One task will be taking longer time. Other tasks will be finished easily.

## During spark application execution which error will indicate spark data might have data skew issue? IMP

breach of MAX\_SIZE limits due to shuffle

## How will you solve data skew issue? IMP

<https://dzone.com/articles/improving-the-performance-of-your-spark-job-on-ske>

* **Partitioning the data in right way** can avoid data skew and get better performance and avoid memory issues and for better resource utilization.

To perform better partitioning I should know size, type and how data is distributed.

* **Repartition the data using appropriate key** which can spread the load evenly is recommended.
* **BroadCastJoin**: Bump up **spark.sql.autoBroadcastJoinThreshold**

spark.sql.autoBroadcastJoinThreshold: Default value 10485760 (10 MB)

Size of table that will be broadcast to all worker nodes when performing a join. By setting this value to -1 broadcasting can be disabled. Note that currently statistics are only supported for Hive Metastore tables where the command ANALYZE TABLE <table Name> COMPUTE STATISTICS no scan has been run.

* **Iterative (Chunked) Broadcast join:**

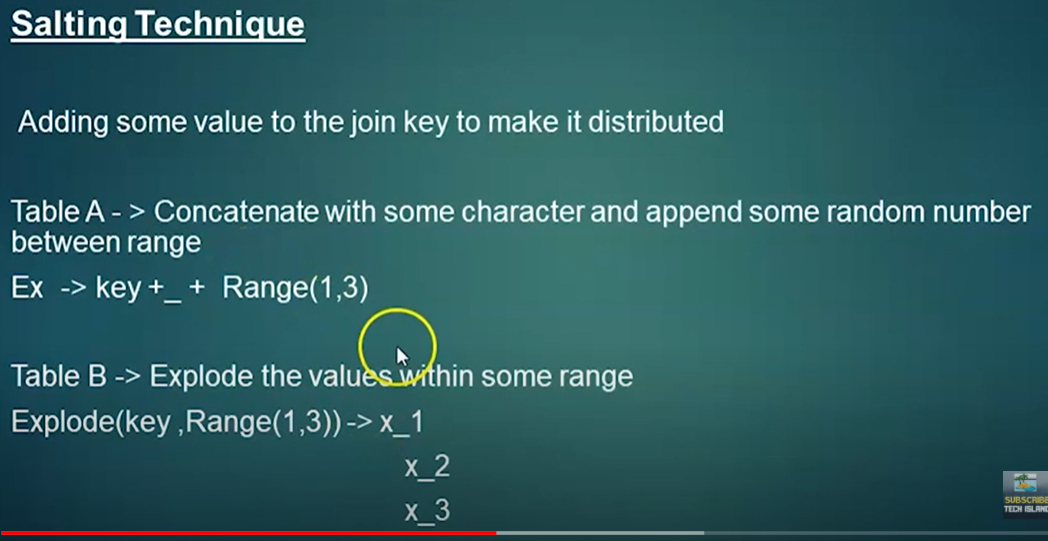
When your smaller table becomes prohibitively large it might be worth considering the approach of iteratively taking slices of your smaller (but not that small) table, broadcasting those, joining with the larger table, then unioning the result. Here is a talk that explains the details nicely.

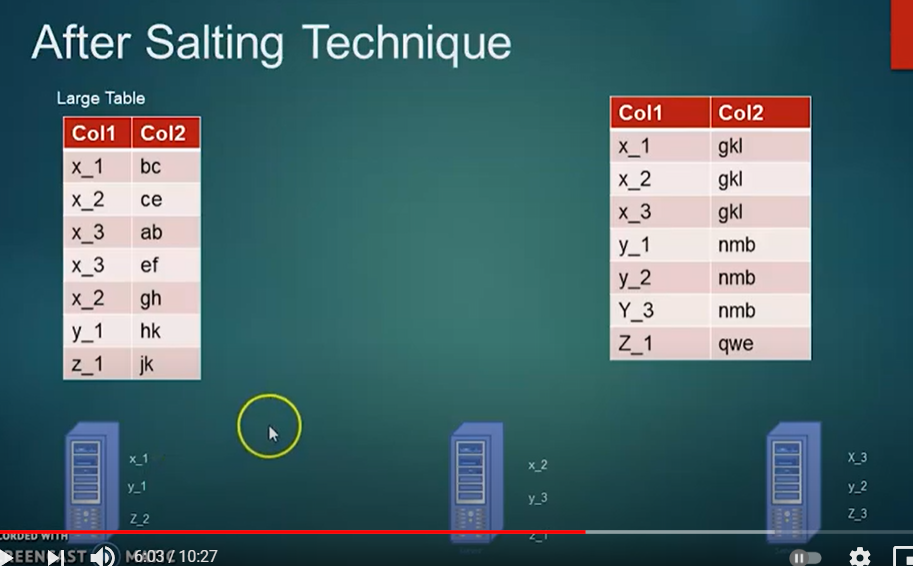
Note: Because broadcast join and iterative broadcast join works well when one of the tables is small. When one of the table is very big and other table is at medium level then it is better to go with salting technique.

* **Salting technique:**

**https://www.youtube.com/watch?v=d41\_X78ojCg**

In the below screen shot table ‘A’ is large table and table ‘B’ is small table.





In case of salting technique, we add **range** of value say 1 to 3 to large table and **explode** the small table by 3

When you are joining medium table we can disable auto broadcast join by setting spark.sql.autoBroadcastJoinThreshold to -1



Understand what is <=> operator in the above code.

* When we setup cluster we need to make sure that all nodes in the cluster should be homogeneous that is all nodes in the cluster should have same configuration in terms of CPU, memory, and disk
* When we setup cluster we need to make sure that all nodes in the cluster should be homogeneous that is all nodes in the cluster should have same configuration in terms of CPU, memory, and disk
* When we setup cluster we need to make sure that all nodes in the cluster should be homogeneous that is all nodes in the cluster should have same configuration in terms of CPU, memory, and disk
* Should try to dedicate Nodes to one service(team) if Node is shared then again, we might face same issue
* There will be techniques provided in each of the technologies like spark sql, hive etc we need to know and apply it properly

# Join

## How join affects performance? IMP

Join is, in general, an expensive operation, so pay attention to the joins in your application to optimize them.

**BroadcastHashJoin** is most performant for cases where one of the relations is small enough that it can be broadcast. Below are some tips:

Join order matters; start with the most selective join. For relations less than spark.sql.autoBroadcastJoinThreshold, you can check whether broadcast HashJoin is picked up.

Use SQL hints if needed to force a specific type of join.

Example: When joining a small dataset with large dataset, a broadcast join may be forced to broadcast the small dataset.

Confirm that Spark is picking up broadcast hash join; if not, one can force it using the SQL hint.

Avoid cross-joins.

Broadcast HashJoin is most performant, but may not be applicable if both relations in join are large.

Collect statistics on tables for Spark to compute an optimal plan.

1. **How to tell if join is skewing in Spark SQL? IMP**

If spark SQL join is taking more time than expected then it could be because of data skewing.

We can get more information by looking in the Spark UI.

We can refer DAG and time each task taking to finish.

<https://coxautomotivedatasolutions.github.io/datadriven/spark/data%20skew/joins/data_skew/>

<https://www.youtube.com/watch?v=bhYV0JOPd9Y>

<https://datarus.wordpress.com/2015/05/04/fighting-the-skew-in-spark/s>

<https://www.youtube.com/watch?t=3215&v=HG2Yd-3r4-M>

1. **Handling data skew adaptively in spark using dynamic partitioning?**

## What is cardinality in database?

https://beginnersbook.com/2015/04/cardinality-in-dbms/

## What is JDBC Data source?

Used to read data from relational databases using JDBC API. This JDBC data source returns data as data frames.

1. **What are the features of Spark SQL?**
2. Integrated
3. Uniform Data Access
4. Hive Compatibility
5. Standard Connectivity
6. Performance & Scalability

# Window Function in Spark SQL

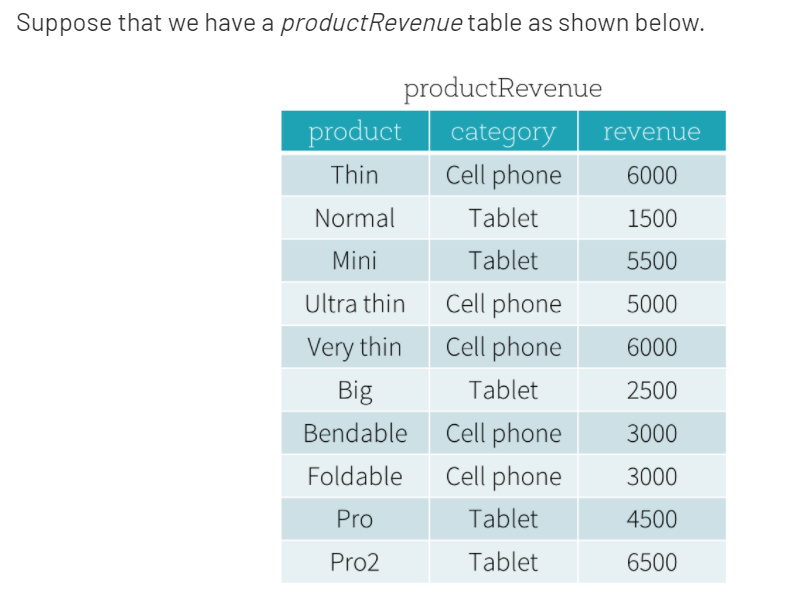
https://databricks.com/blog/2015/07/15/introducing-window-functions-in-spark-sql.html

## What are window functions?

Helps to calculate one result per row based on a window of rows.

It helps us to **calculate moving average, calculating a cumulative sum, or accessing the values of a row appearing before the current row**

## Window function explanation with Example?



We want to answer two questions:

* What are the best-selling and the second best-selling products in every category?
* What is the difference between the revenue of each product and the revenue of the best-selling product in the same category of that product?

## How will you calculate moving average and cumulative sum in spark SQL?

Using window functions.

We can easily calculate moving average or cumulative sum, or we can also reference a value in the previous column

## How can you reference value of previous row of a table in spark?

Using Window functions.

Note: Window functions are complementary to existing Data Frame operations.

# Important Spark configurations

# 