Database Partitioning

Database Partitioning **1**
- Intro 1
- Horizontal partitioning / Sharding 2
  - Partitioning criteria 3
- Vertical partitioning 3
- Case Study: Apache Cassandra 4
- Case Study: Apache Spark 5
- * References 6

Intro

Consider partitioning as a **fundamental** part of system design even if the system initially only contains a single partition.

A **partition** is a division of a logical database or its constituent elements into distinct **independent** parts.

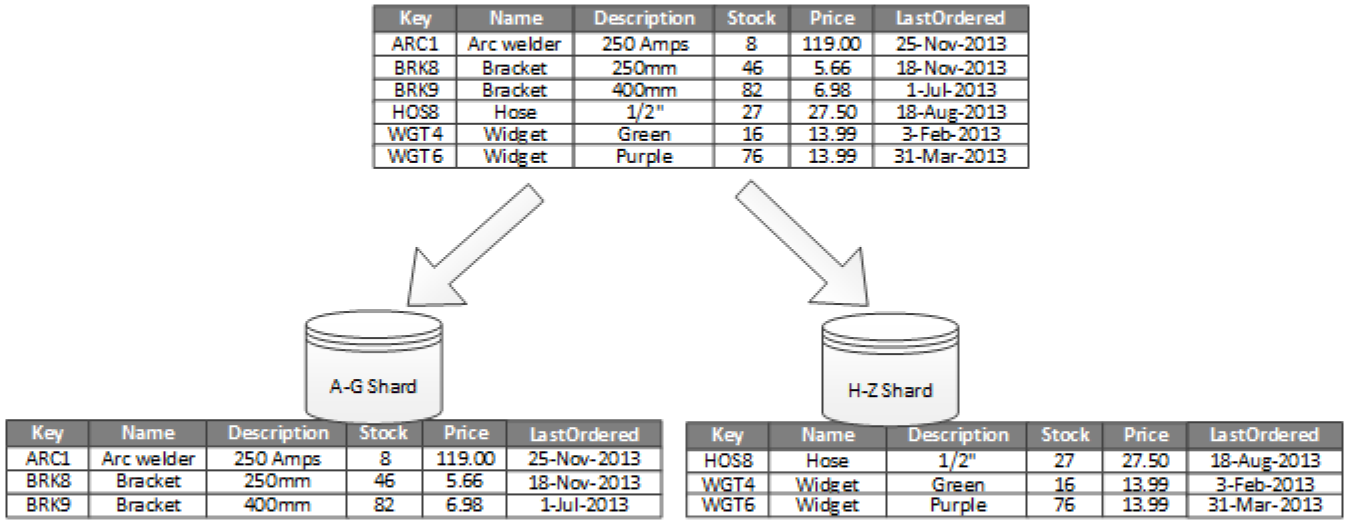
Partitioning is normally done for **manageability**, **performance**, **availability and load balancing**.

It is popular in **distributed database management systems**.

Horizontal partitioning / Sharding

Horizontal partitioning / Sharding involves putting different **rows** into different tables.

Below is an example of horizontal partitioning.



Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013

Key	Name	Description	Stock	Price	LastOrdered
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Product inventory data is divided into shards based on the product **key**.

Each shard holds the data for a contiguous **range** of shard keys (A-G and H-Z), organized alphabetically.

The sharding key must ensure that data is partitioned to spread the workload as **evenly** as possible across the shards.

The shards don't have to be the same size. It's more important to balance the **number of requests**.

In practice, we will need to repartition: split large shards, coalesce small shards into larger partitions, or change the schema.

These operations are **time consuming**, and might require taking one or more shards **offline** while they are performed. Replication comes into play.

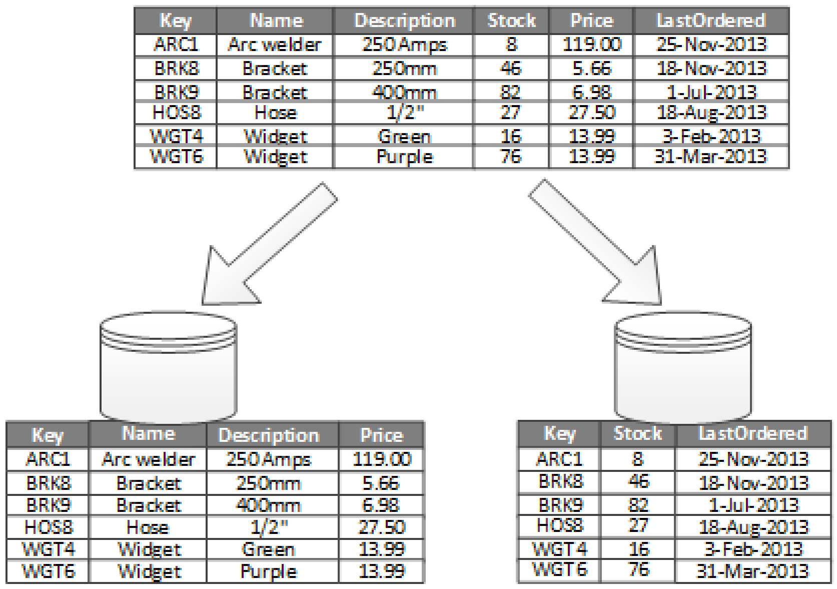
Partitioning criteria

	Definition	Explanation
Range partitioning	A partition is selected based on column values falling **within** a given **range**.	e.g. A partition for all rows where the column Age has a value between 0 and 18.
List partitioning	A partition is selected based on column values **matching one of** a set of discrete values.	e.g. A partition for all rows where the column City is either Palo Alto, Mountain View, Sunnyvale or San Jose, etc for Santa Clara County.
Round-robin partitioning	With n partitions, the ith row in insertion order is assigned to the partition **(i mod n)**.	Pros: sequential access to a relation to be done in parallel Cons: direct access to individual row, based on a predicate, requires accessing the entire relation
Hash partitioning	Applies **a hash function** to some attribute that yields the partition number	
Composite partitioning	Combinations of the above.	e.g. Consistent hashing could be considered a composite of hash and list partitioning where the hash reduces the key space to a size that can be listed.

Vertical partitioning

Vertical partitioning involves putting different **columns** into different tables.

Below is an example of vertical partitioning.



Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Price
ARC1	Arc welder	250 Amps	119.00
BRK8	Bracket	250mm	5.66
BRK9	Bracket	400mm	6.98
HOS8	Hose	1/2"	27.50
WGT4	Widget	Green	13.99
WGT6	Widget	Purple	13.99

Key	Stock	LastOrdered
ARC1	8	25-Nov-2013
BRK8	46	18-Nov-2013
BRK9	82	1-Jul-2013
HOS8	27	18-Aug-2013
WGT4	16	3-Feb-2013
WGT6	76	31-Mar-2013

One partition holds Name, Description, Price, which are slow-moving and accessed more frequently.

The other holds Stock, LastOrdered, which are dynamic, commonly used together, and sensitive.

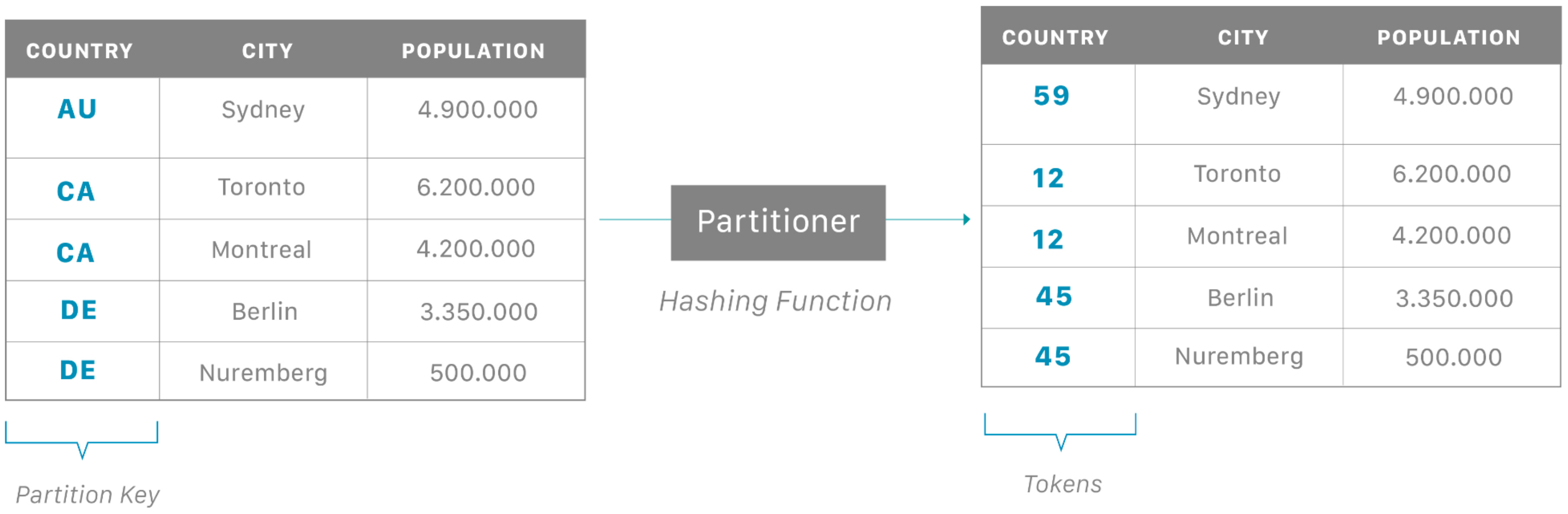
Case Study: Apache Cassandra

Cassandra is a horizontally-partitioned row store.

In Cassandra, each **node** owns a range of **tokens**.

When data is inserted into the cluster, the first step is to apply a **hash** function to the **partition key**.

The output **token** is used to determine what node (based on the token **range**) will get the data.



COUNTRY	CITY	POPULATION
AU	Sydney	4.900.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
DE	Berlin	3.350.000
DE	Nuremberg	500.000

COUNTRY	CITY	POPULATION
59	Sydney	4.900.000
12	Toronto	6.200.000
12	Montreal	4.200.000
45	Berlin	3.350.000
45	Nuremberg	500.000

Cassandra uses **consistent hashing**: it maps every node to one or more **tokens on** a continuous **hash ring**, and defines ownership by hashing a key onto the ring and then "walking" the ring in one direction.

The main difference of consistent hashing to naive data hashing is that when the **number of nodes** to hash into **changes**, consistent hashing only has to **move a small fraction of the keys**.

Case Study: Apache Spark

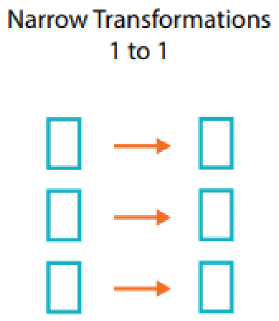
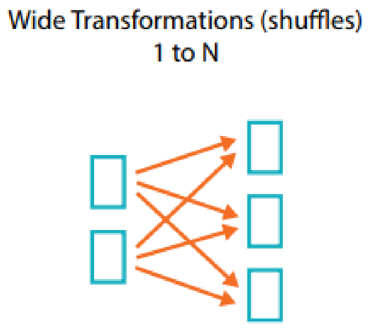
Spark is a distributed computing engine. Its low-level data abstraction is a **resilient** distributed dataset (RDD). Resilient because RDDs are **immutable** (can't be modified once created) and **fault tolerant.**

RDD is a distributed **collection of objects**, which are **stored in partition**s.

Hash partitioning, range partitioning and custom spark partitioning are supported. Operations such as groupByKey, reduceByKey and sort automatically result in a hash or range partitioned RDD.

Spark creates one task per partition, if there are too many partitions, most of the time goes into creating, scheduling, and managing the tasks then executing. If there are too few partitions, cluster resources may not be fully utilized due to less parallelism.

Relations between partitions of RDDs are categorized as below:

Narrow dependency	**Wide dependencies**
each partition of the parent RDD is used by at most one partition of the child RDD	each partition of the parent RDD is used by multiple partitions of the child RDD
pipelined execution on one node	shuffled across nodes
	

We may repartition wisely to

* References

https://docs.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning
https://www.digitalocean.com/community/tutorials/understanding-database-sharding
https://en.wikipedia.org/wiki/Partition_(database)
https://github.com/apache/cassandra
https://cassandra.apache.org/_/index.html
https://techmagie.wordpress.com/2015/12/19/understanding-spark-partitioning/
https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf