Partitioning Tables for Faster Queries

Overview

Optimize queries with Hive partitions

Work with partitioned tables

- Creating partitions
- Loading data into partitions
- Querying data from partitions

Implement static and dynamic partitioning on Hive tables

Faster Queries on Traditional Databases

Designing Databases

Faster queries

Retrieving information from a database should be fast

Designing Databases

Traditional database admins design data layouts based on usage patterns

Usage Patterns and Design

Do users access customer and order information together?

Customer id is a foreign key in the orders table

Create a Customer_Orders table grouping both information together

Usage Patterns and Design

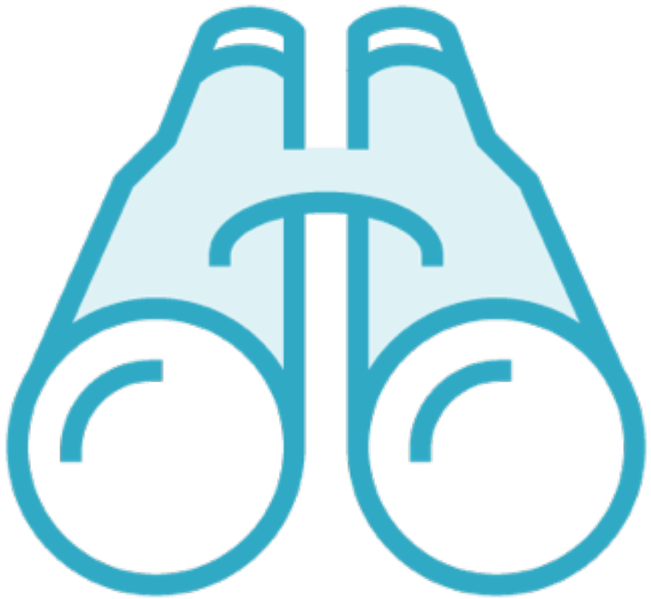
Do users often require order information for a particular date?

Create an index on the date column of the orders table

Indexes

Faster lookup

Speed up data retrieval in traditional relational databases

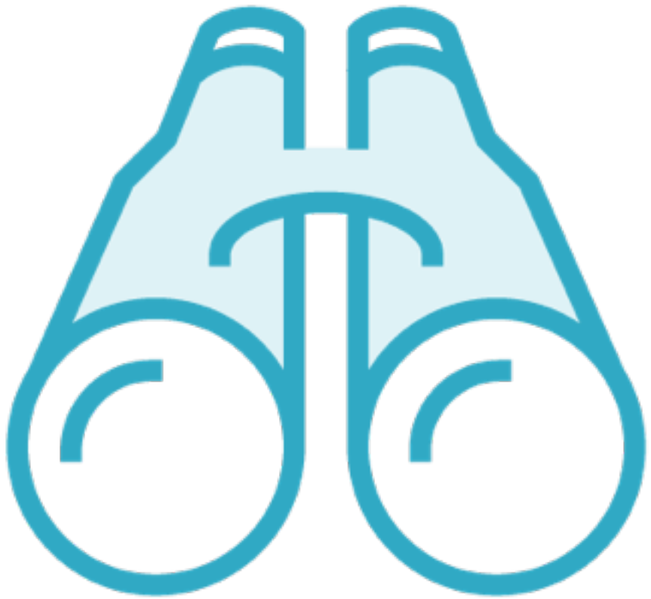


Indexes

```
select * from Orders
where date = “12/07/2016”
```

```
select * from Orders
where date > “12/07/2016”
and date < “16/07/2016”
```

Indexes

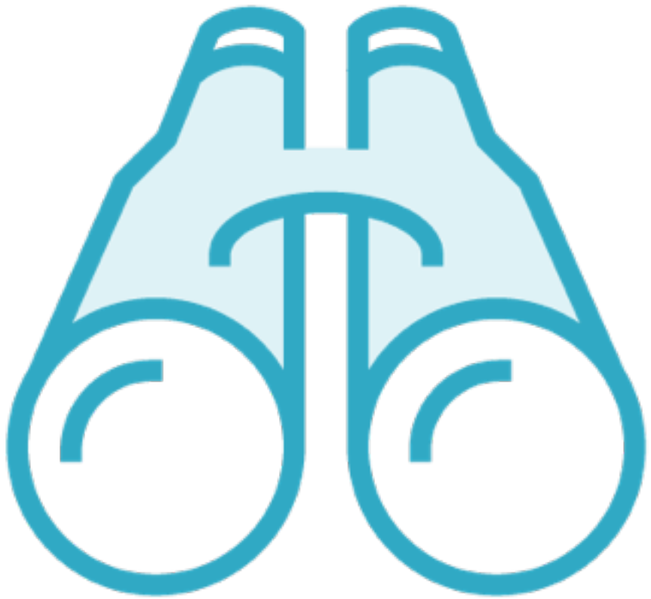


Associated with a table which allow fast lookup of records

Created from one or more columns

Copy of column data that can be searched very efficiently

Indexes



Well-designed indexes are can be a major performance improvement for queries

Indexes ~ Partitioning

Partitions are the logical equivalent of indexes in Hive

Partitioning Tables in Hive

Partitioning

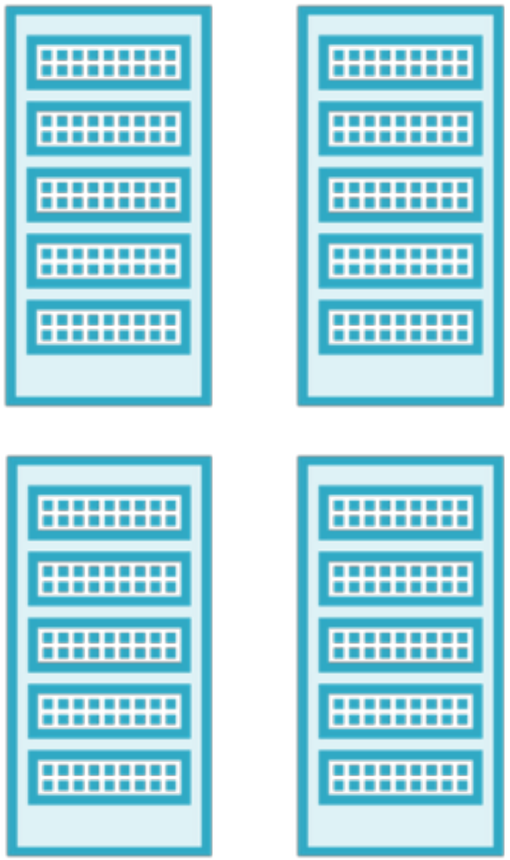
Split data into smaller subsets

Separate records into manageable parts based on a column value

Partitioning

Consider an e-commerce site with order data from across the United States

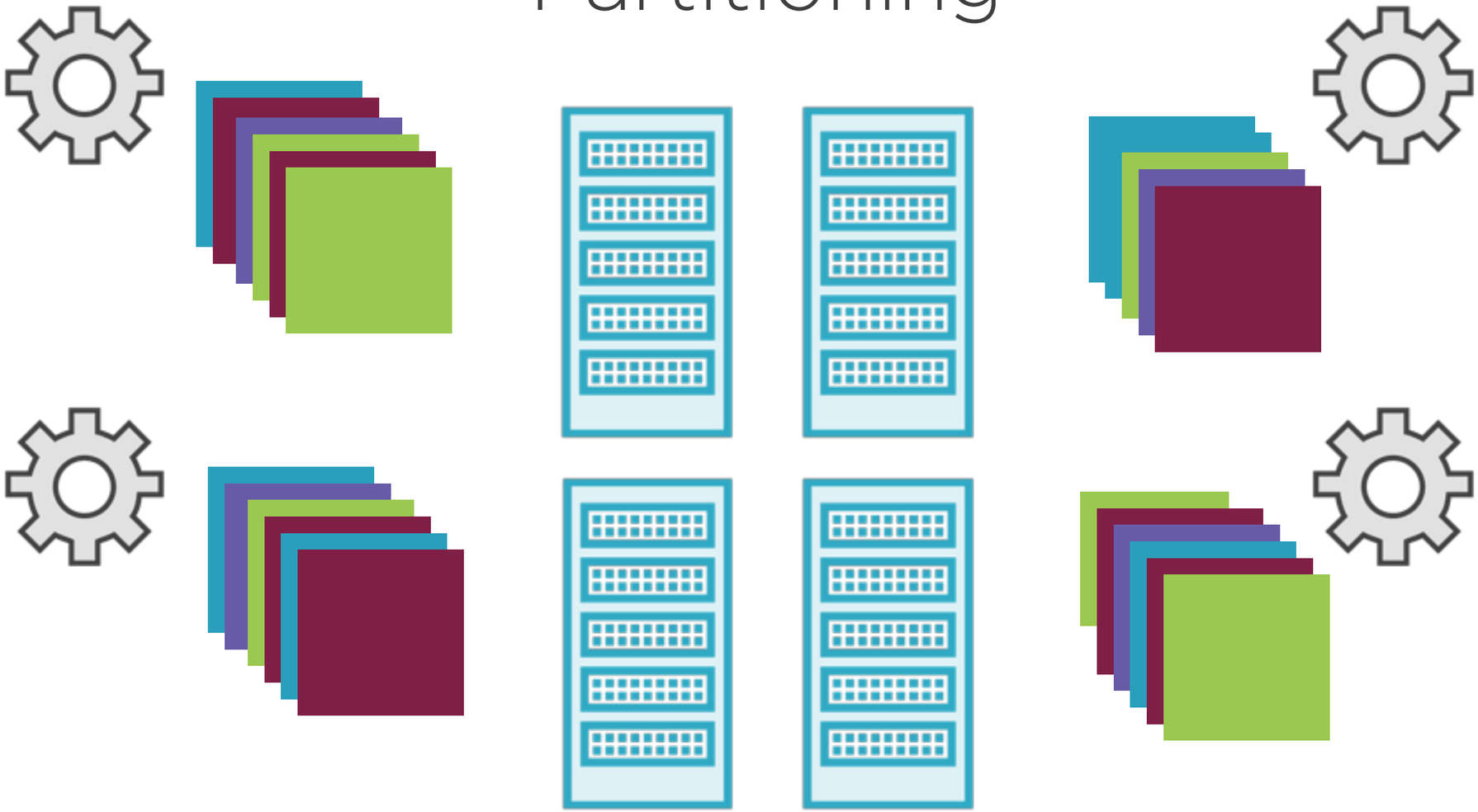
Partitioning



Partitioning

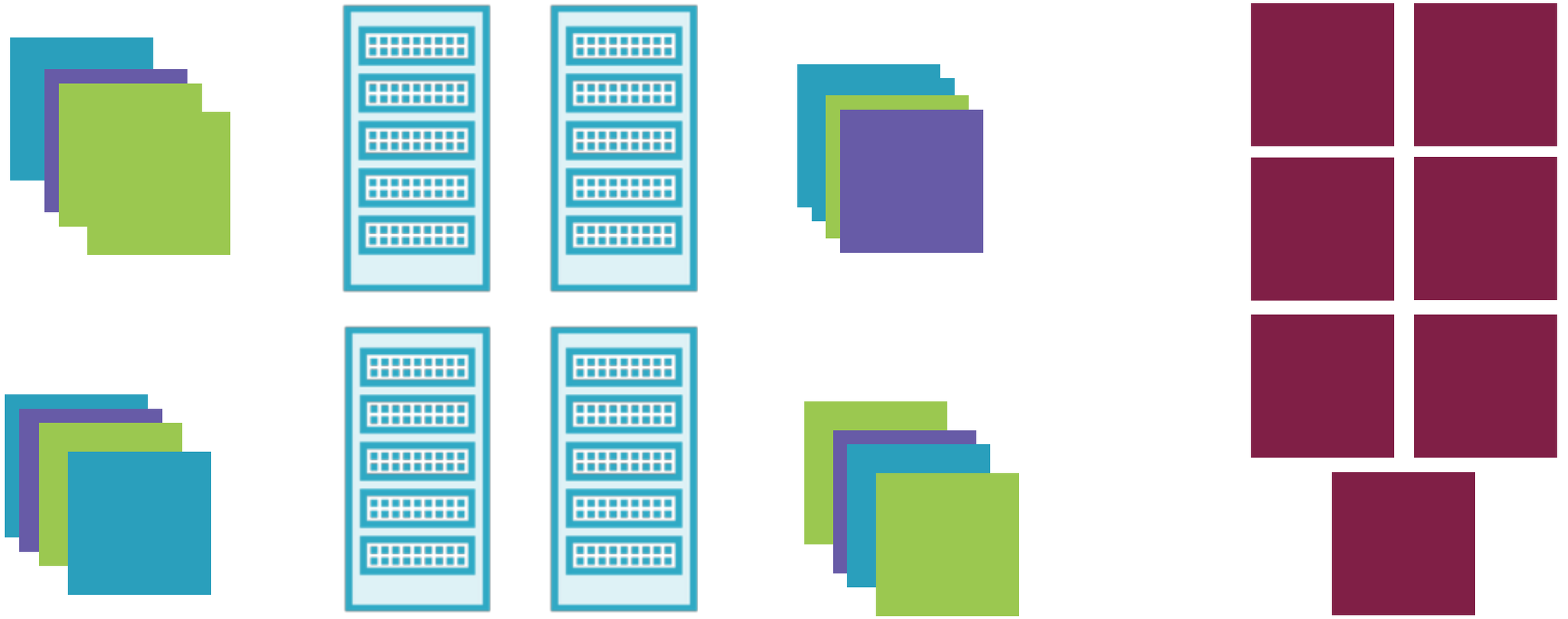
Records will be split across multiple machines in the cluster

Partitioning



select * from Orders where state = “WA”

Partitioning



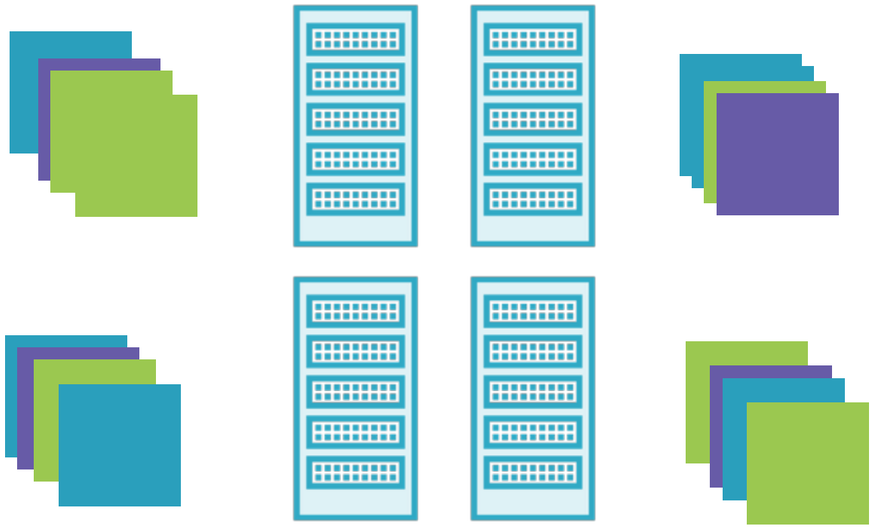
select * from Orders where state = “WA”

Partitioning

All records on each machine need to be scanned to retrieve results

select * from Orders where state = “WA”

Partitioning



select * from Orders where state = “WA”

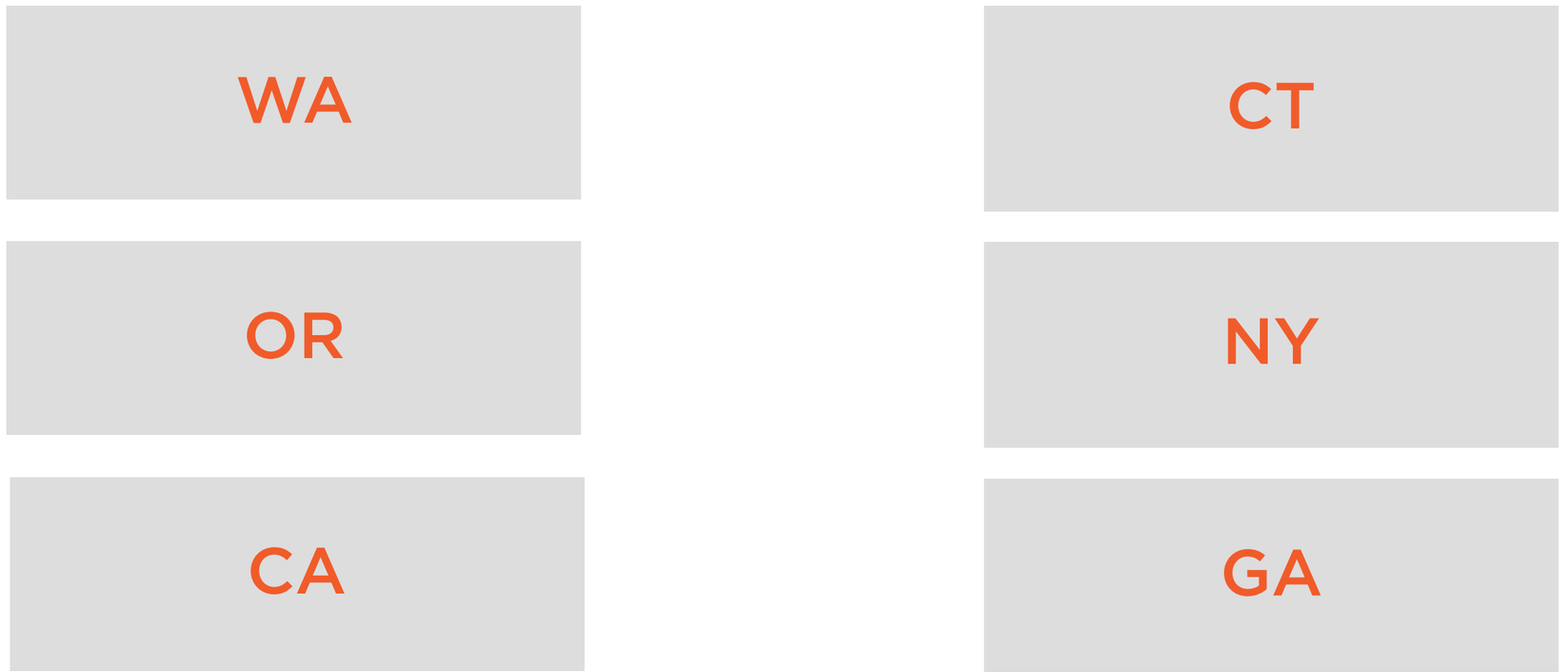
What if the dataset is huge on each node?

What if this is the most common query run?

Partitioning

Data may be naturally split into logical units

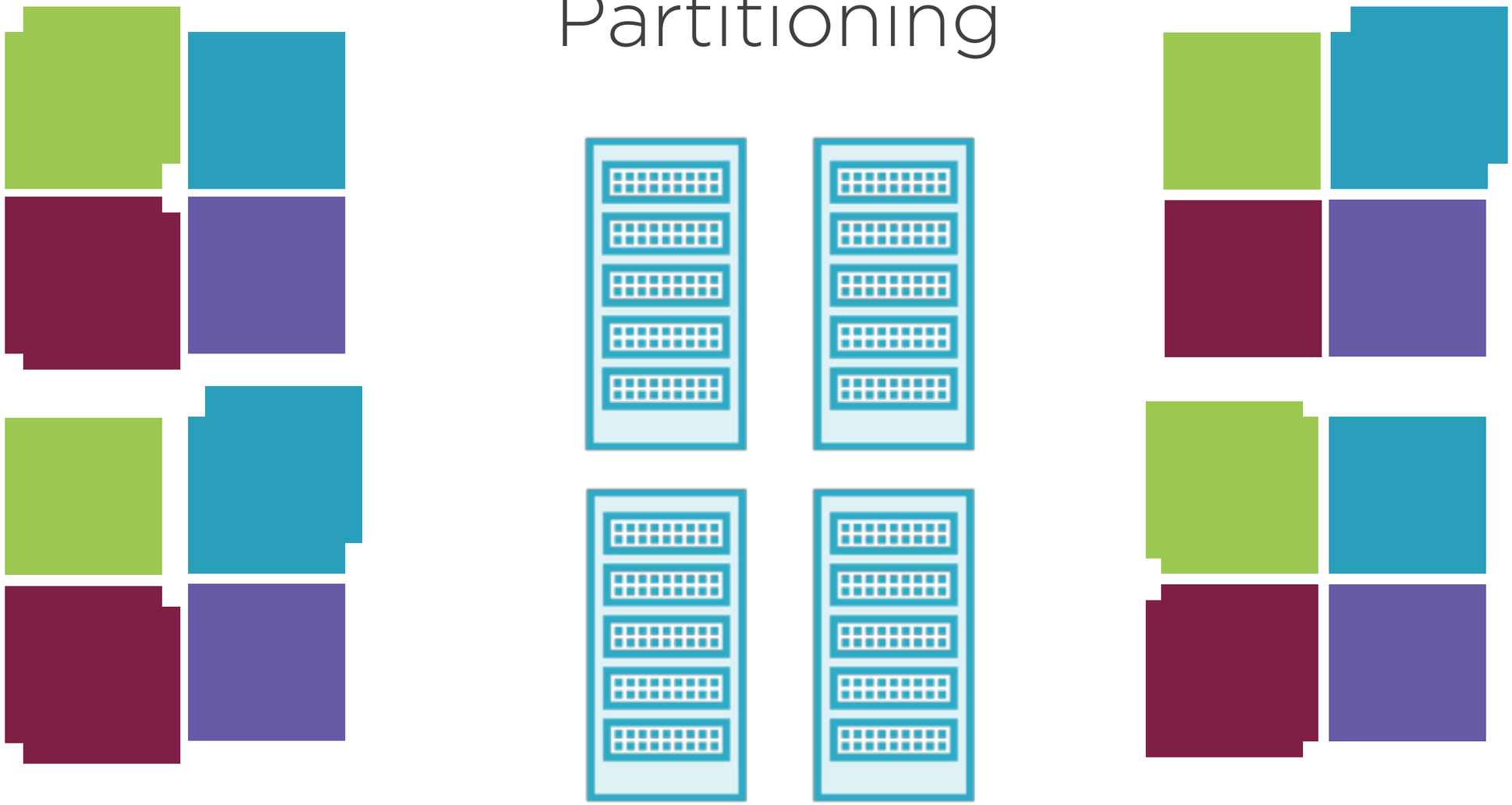
Partitioning



Partitioning

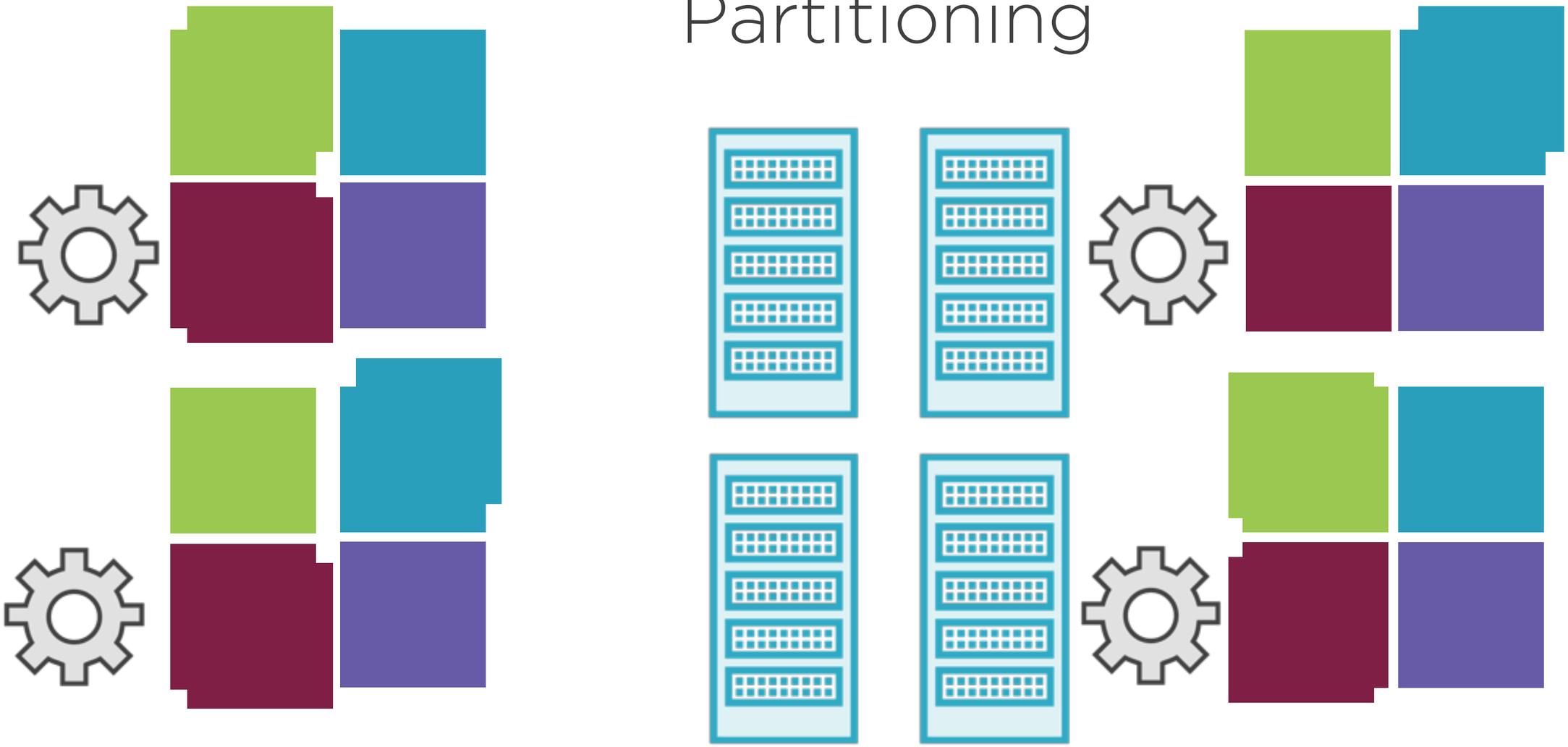
Partition this data on state information

Partitioning



Records from each state will be in a different directory

Partitioning



State specific queries will run only on data in one directory

Partitioning

The records in a Hive table are typically stored in HDFS

by default in /user/hive/warehouse

Partitioning

/user/hive/warehouse/db/orders

/state=CA

/state=NY

/state=WA

/state=NJ

Partitioning

/user/hive/warehouse/db/orders

/state=CA

/file-01

/file-02

/file-03

Partitions vs. Indexes

Partitions	Indexes
Physical storage of records affected	No change to how the actual records are stored
No additional data structures used for lookup	Additional data structures hold a copy of indexed column values
Huge datasets	Relatively smaller datasets

Demo

Create a managed partitioned table

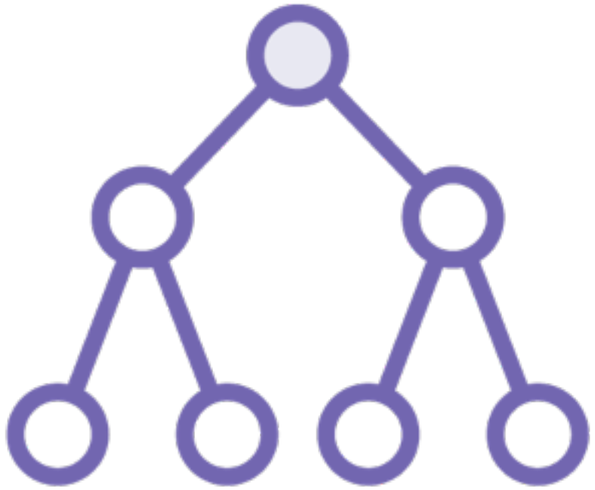
Load data into a partitioned table

Query the partitioned table

See how partitioned tables are laid out in HDFS

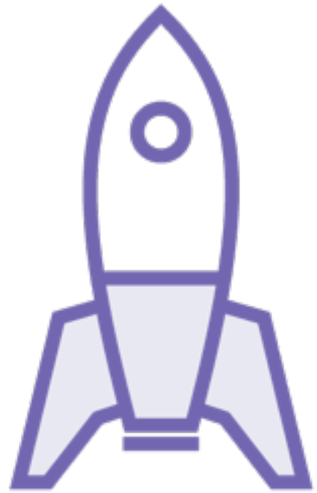
Why Partition Tables?

Why Partition Tables?



Logical organization of data

Set up directories and sub-directories in Hive based on how data is partitioned



Improve query performance

Only scan the data in the partition where the result might be found

Why Partition Tables?

ID	Customer	Product	Quantity	Amount	State
c1	John	iPhone	1	599	CA
c2	Jill	Doll	2	58	WA
c3	Emily	Shoes	1	66	NJ
c4	Nina	Jeans	1	99	NJ
c5	Rick	Skates	1	199	CA
c6	Valerie	Make Up	1	88	WA
c7	Olga	Book	3	45	NY
c8	Steven	Belt	2	25	NY

Orders

Why Partition Tables?

state=CA

ID	Customer	Product	Quantity	Amount	State
c1	John	iPhone	1	599	CA
c5	Rick	Skates	1	199	CA

state=WA

ID	Customer	Product	Quantity	Amount	State
c2	Jill	Doll	2	58	WA
c6	Valerie	Make	1	88	WA

state=NJ

ID	Customer	Product	Quantity	Amount	State
c3	Emily	Shoes	1	66	NJ
c4	Nina	Jeans	1	99	NJ

state=NY

ID	Customer	Product	Quantity	Amount	State
c7	Olga	Book	3	45	NY
c8	Steven	Belt	2	25	NY

```
select * from orders
where state = “NJ”
and product = “Jeans”
```

Why Partition Tables?

state=NJ

ID	Customer	Product	Quantity	Amount	State
c3	Emily	Shoes	1	66	NJ
c4	Nina	Jeans	1	99	NJ

```
select * from orders
where state = “NJ”
and product = “Jeans”
```

Only one partition needs to be scanned

Why Partition Tables?

state=NJ

ID	Customer	Product	Quantity	Amount	State
c3	Emily	Shoes	1	66	NJ
c4	Nina	Jeans	1	99	NJ

```
select * from orders
where state = “NJ”
and product = “Jeans”
```

Potentially huge performance gains

Why Partition Tables?

state=CA

ID	Customer	Product	Quantity	Amount	State
c1	John	iPhone	1	599	CA
c5	Rick	Skates	1	199	CA

state=WA

ID	Customer	Product	Quantity	Amount	State
c2	Jill	Doll	2	58	WA
c6	Valerie	Make	1	88	WA

state=NJ

ID	Customer	Product	Quantity	Amount	State
c3	Emily	Shoes	1	66	NJ
c4	Nina	Jeans	1	99	NJ

state=NY

ID	Customer	Product	Quantity	Amount	State
c7	Olga	Book	3	45	NY
c8	Steven	Belt	2	25	NY

```
select * from orders
where quantity > 1
and product = “Jeans”
```

Why Partition Tables?

state=CA

ID	Customer	Product	Quantity	Amount	State
c1	John	iPhone	1	599	CA
c5	Rick	Skates	1	199	CA

state=WA

ID	Customer	Product	Quantity	Amount	State
c2	Jill	Doll	2	58	WA
c6	Valerie	Make	1	88	WA

state=NJ

ID	Customer	Product	Quantity	Amount	State
c3	Emily	Shoes	1	66	NJ
c4	Nina	Jeans	1	99	NJ

state=NY

ID	Customer	Product	Quantity	Amount	State
c7	Olga	Book	3	45	NY
c8	Steven	Belt	2	25	NY

```
select * from orders
where quantity > 1
and product = “Jeans”
```

All partitions need to be scanned

Why Partition Tables?

state=CA

ID	Customer	Product	Quantity	Amount	State
c1	John	iPhone	1	599	CA
c5	Rick	Skates	1	199	CA

state=WA

ID	Customer	Product	Quantity	Amount	State
c2	Jill	Doll	2	58	WA
c6	Valerie	Make	1	88	WA

state=NJ

ID	Customer	Product	Quantity	Amount	State
c3	Emily	Shoes	1	66	NJ
c4	Nina	Jeans	1	99	NJ

state=NY

ID	Customer	Product	Quantity	Amount	State
c7	Olga	Book	3	45	NY
c8	Steven	Belt	2	25	NY

Partition should be based on the most common queries run

Use **partitions** intelligently for the most **common** queries

Demo

Load data into a managed partitioned table from files

Demo

Working with an external partitioned table

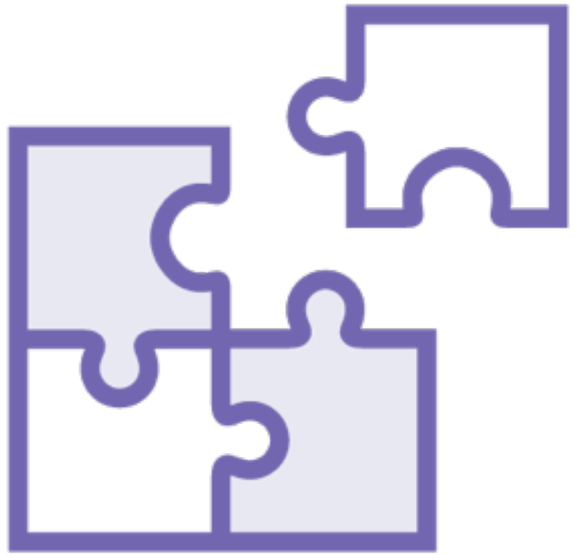
- add partitions using the alter command

- change partitions once created

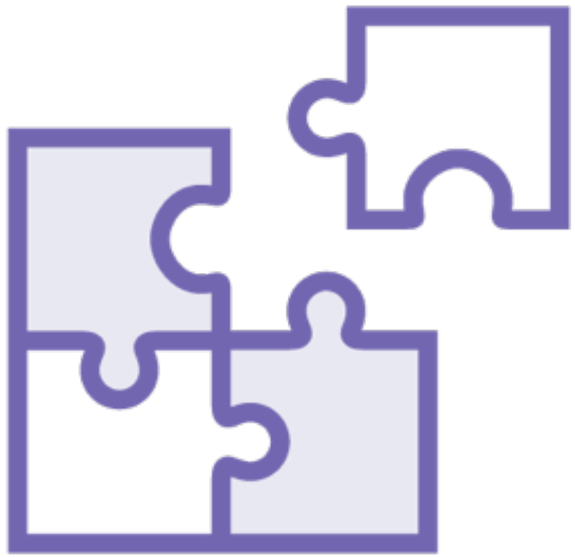
- drop partitions

Partitioning Trade-offs

Partitioning Trade-offs



How many partitions should you have?



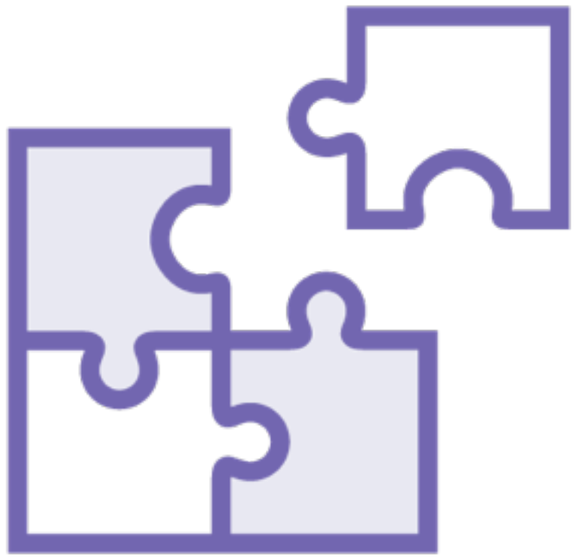
Large Number of Partitions

Partition on customer id on the orders table?

Millions of partitions, an HDFS directory for each partition

Huge overhead for the NameNode in Hadoop

May optimize some queries but be detrimental for others



Small Number of Partitions

Partition on product quantity on the orders table?

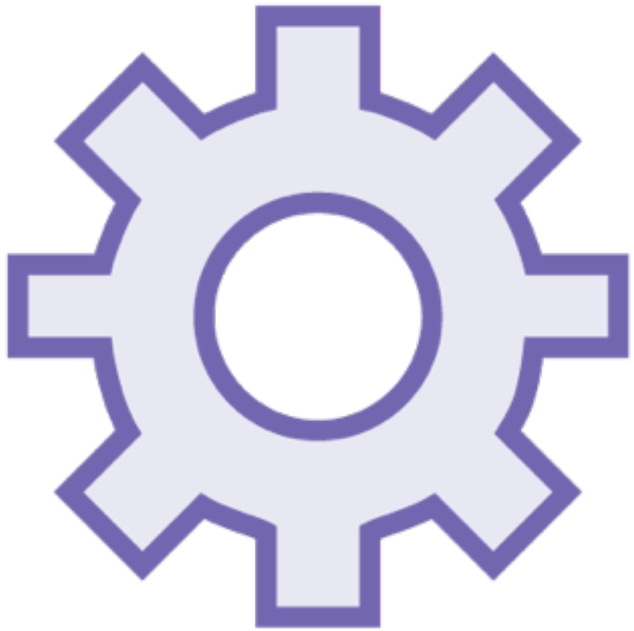
Very few partitions, few logical groupings in our data

No real optimizations on queries run

No reduction in the data scanned by queries

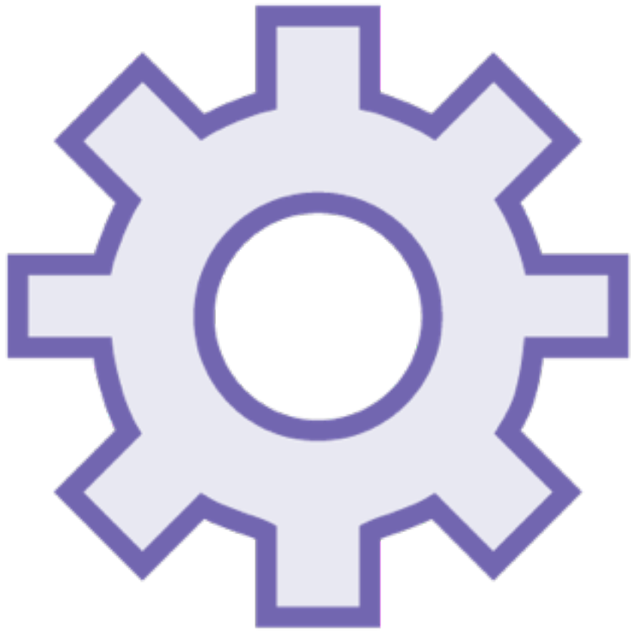
Static and Dynamic Partitioning

Static Partitioning



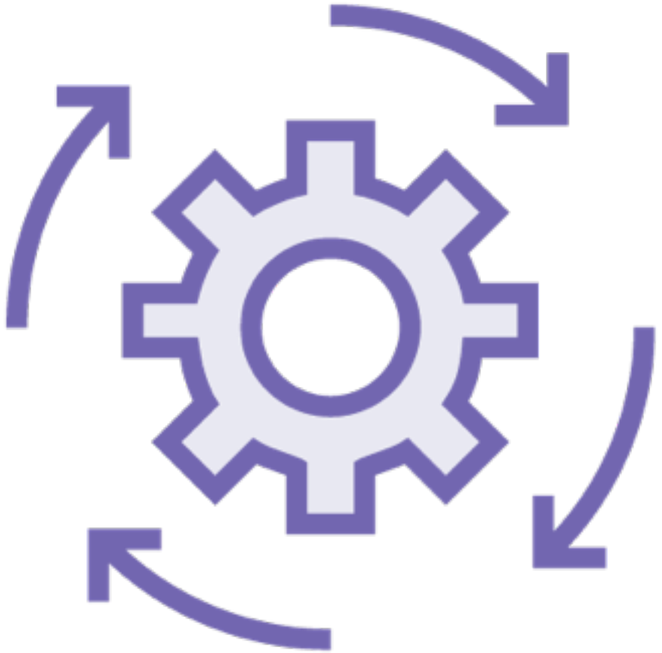
Partitions we've seen so far have been static partitions

Static Partitioning



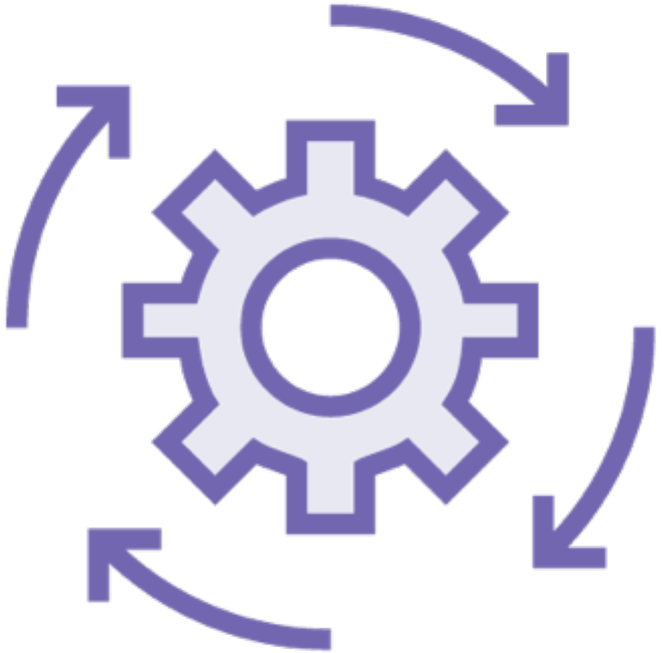
Managing partitions manually is not a scalable process

Dynamic Partitioning



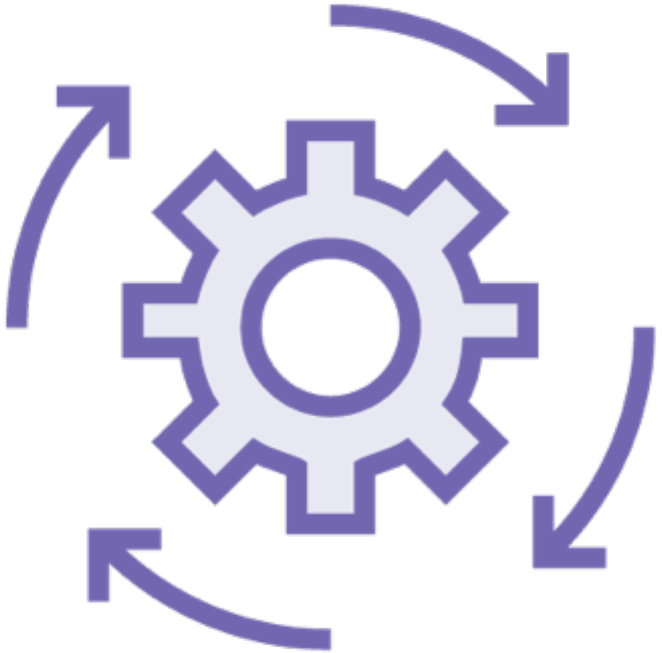
Dynamic partitioning allows Hive to automatically create partitions on data

Dynamic Partitioning



Partitions are based on the values of the partition keys

Dynamic Partitioning



Properties to be set in hive-site.xml or on a per-session basis

```
set hive.exec.dynamic.partition=true
```

Enable Dynamic Partitioning

The default for Hive is disabled, need to explicitly enable dynamic partitioning

```
set hive.exec.dynamic.partition.mode=nonstrict
```

All Partitions Can Be Dynamic

Default is the strict mode where there should be at least one static partition

Non-strict allows all partitions to be dynamic

```
set hive.exec.max.dynamic.partitions=1000
```

Dynamic Partitioning Hogs Resources

Total number of dynamic partitions created

If exceeded, an exception is raised at the end of the job

```
set hive.exec.max.dynamic.partitions.pernode=3
```

Dynamic Partitioning Hogs Resources

The number of dynamic partitions created by each mapper or reducer

If exceeded, a fatal error will be thrown and the job will be killed

```
set hive.exec.max.created.files=150000
```

Dynamic Partitioning Hogs Resources

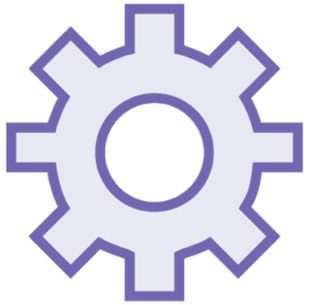
The maximum number of files created by all mappers and reducers

If exceeded, a fatal error is thrown

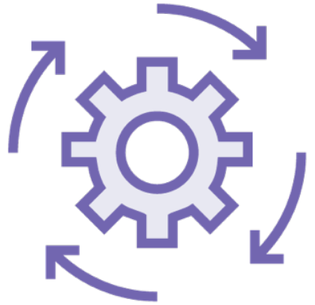
Demo

Set the properties to enable dynamic partitioning in Hive

Create and load a table with dynamic partitioning



Static and Dynamic Partitioning



Static	Dynamic
Partition column values are known before loading data	Partition column values known only when data is loaded
User needs to manually load data into partitions	Hive automatically loads data into partitions
Used when partitions are known and the directory structure exists in HDFS	Used when loading from an existing table that is not partitioned
Fine grained control over exact partition values	Less control over the exact partition values

Demo

Create and load a table partitioned on more than one column

Summary

Organized data and optimized queries with Hive partitions

Worked with managed and external partitioned tables

Implemented static and dynamic partitioning and understood when each should be used