Data lake performance

Limiting data exchanges

Starburst

Lesson objectives

Data lake performance: Limit data exchanges

1. Understand the use case being used for lab exercises.

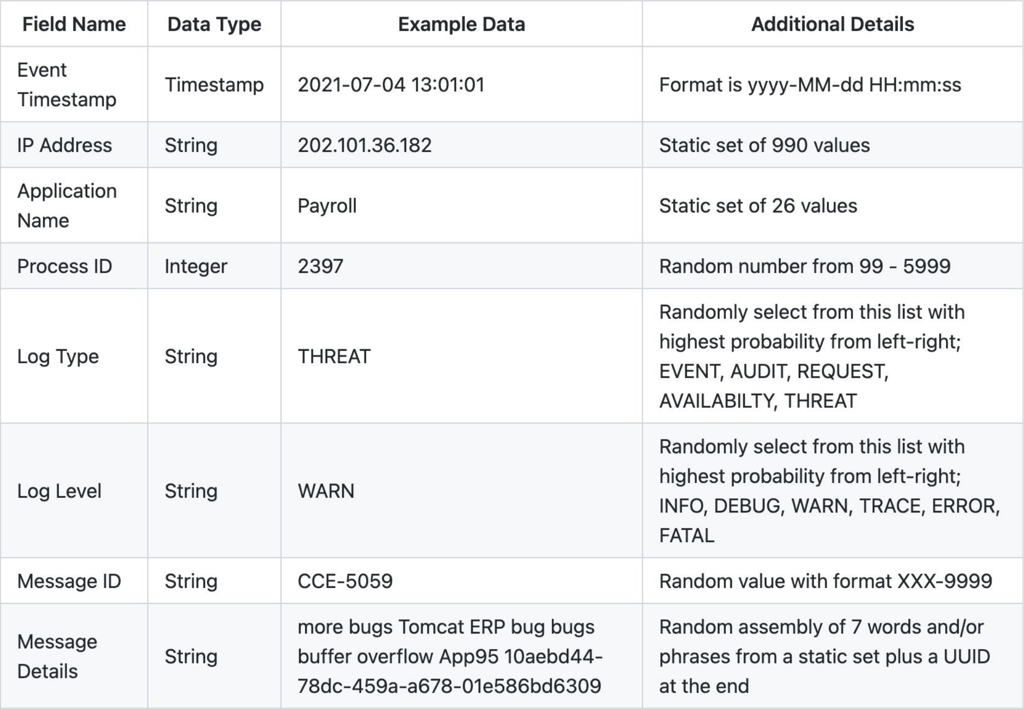
2. Explain the impact that linear scalability has on Starburst clusters.

3. Describe the role that projection and filtering play on queries, and the SQL commands needed to execute them.

4. Describe how data exchanges work, and the impact they have on performance.

Lab dataset: Server logs

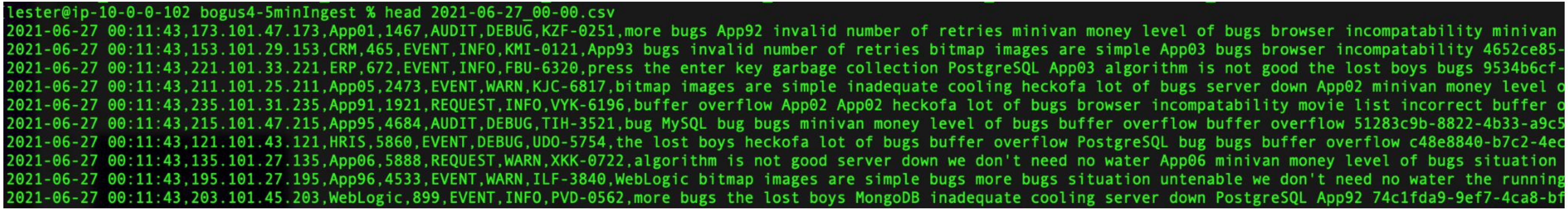
Data format



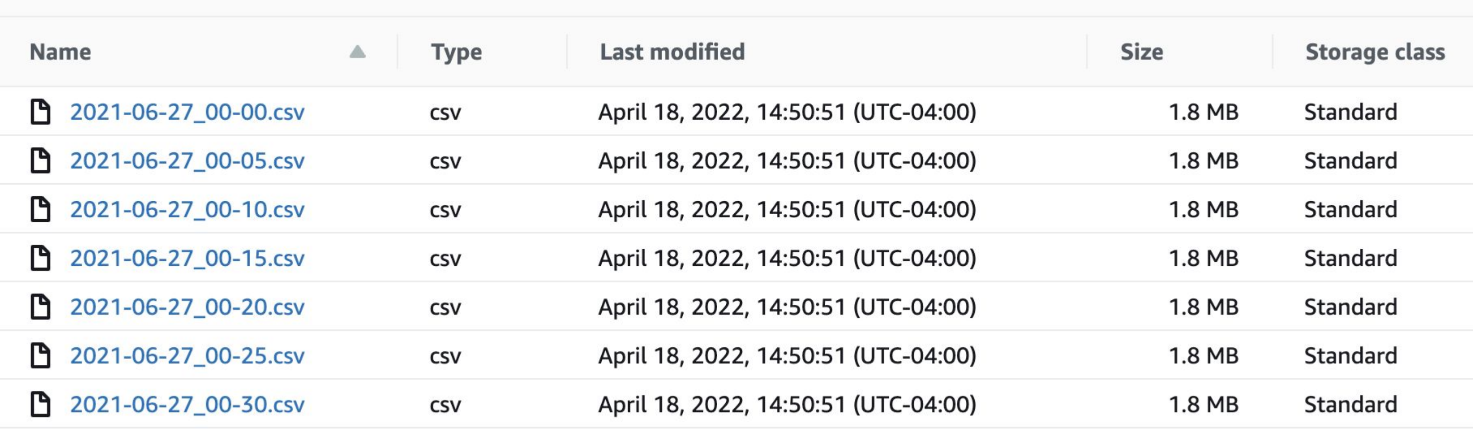
Field Name	Data Type	Example Data	Additional Details
Event Timestamp	Timestamp	2021-07-04 13:01:01	Format is yyyy-MM-dd HH:mm:ss
IP Address	String	202.101.36.182	Static set of 990 values
Application Name	String	Payroll	Static set of 26 values
Process ID	Integer	2397	Random number from 99 - 5999
Log Type	String	THREAT	Randomly select from this list with highest probability from left-right; EVENT, AUDIT, REQUEST, AVAILABILTY, THREAT
Log Level	String	WARN	Randomly select from this list with highest probability from left-right; INFO, DEBUG, WARN, TRACE, ERROR, FATAL
Message ID	String	CCE-5059	Random value with format XXX-9999
Message Details	String	more bugs Tomcat ERP bug bugs buffer overflow App95 10aebd44-78dc-459a-a678-01e586bd6309	Random assembly of 7 words and/or phrases from a static set plus a UUID at the end

Lab artifact - Log Data Generator (GitHub)

Lab dataset: Load into data lake



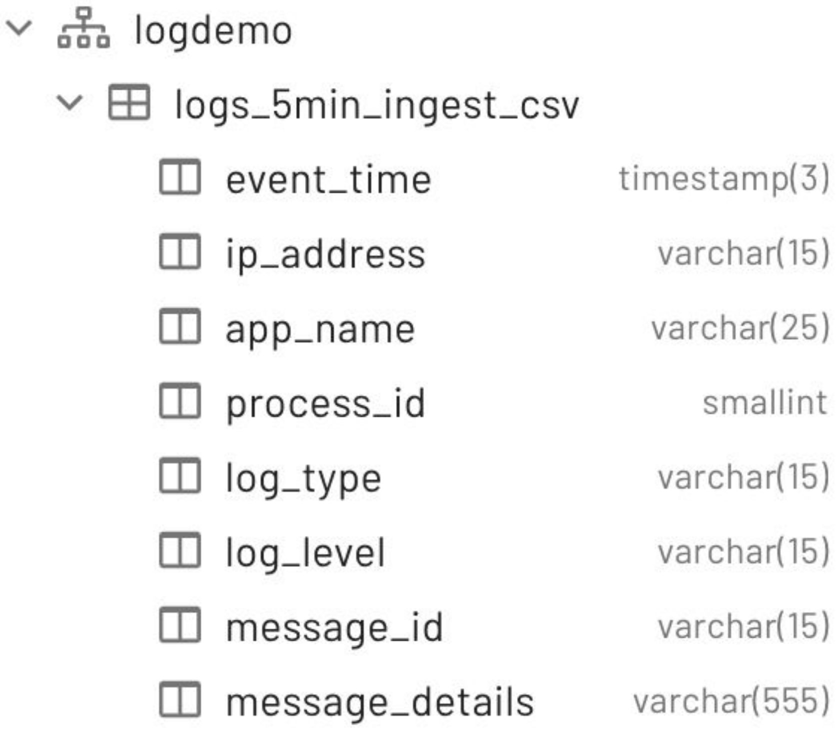
```
lester@ip-10-0-0-102 bogus4-5minIngest % head 2021-06-27_00-00.csv
2021-06-27 00:11:43,173.101.47.173,App01,1467,AUDIT,DEBUG,KZF-0251,more bugs App92 invalid number of retries minivan money level of bugs browser incompatability minivan
2021-06-27 00:11:43,153.101.29.153,CRM,465,EVENT,INFO,KMI-0121,App93 bugs invalid number of retries bitmap images are simple App03 bugs browser incompatability 4652ce85-
2021-06-27 00:11:43,221.101.33.221,ERP,672,EVENT,INFO,FBU-6320,press the enter key garbage collection PostgreSQL App03 algorithm is not good the lost boys bugs 9534b6cf-
2021-06-27 00:11:43,211.101.25.211,App05,2473,EVENT,WARN,KJC-6817,bitmap images are simple inadequate cooling heckofa lot of bugs server down App02 minivan money level o
2021-06-27 00:11:43,235.101.31.235,App91,1921,REQUEST,INFO,VYK-6196,buffer overflow App02 App02 heckofa lot of bugs browser incompatability movie list incorrect buffer o
2021-06-27 00:11:43,215.101.47.215,App95,4684,AUDIT,DEBUG,TIH-3521,bug MySQL bug bugs minivan money level of bugs buffer overflow buffer overflow 51283c9b-8822-4b33-a9c5
2021-06-27 00:11:43,121.101.43.121,HRIS,5860,EVENT,DEBUG,UDO-5754,the lost boys heckofa lot of bugs buffer overflow PostgreSQL bug bugs buffer overflow c48e8840-b7c2-4ec
2021-06-27 00:11:43,135.101.27.135,App06,5888,REQUEST,WARN,XKK-0722,algorithm is not good server down we don't need no water App06 minivan money level of bugs situation
2021-06-27 00:11:43,195.101.27.195,App96,4533,EVENT,WARN,ILF-3840,WebLogic bitmap images are simple bugs more bugs situation untenable we don't need no water the running
2021-06-27 00:11:43,203.101.45.203,WebLogic,899,EVENT,INFO,PVD-0562,more bugs the lost boys MongoDB inadequate cooling server down PostgreSQL App92 74c1fda9-9ef7-4ca8-bf
```



Name ▲	Type	Last modified	Size	Storage class
2021-06-27_00-00.csv	csv	April 18, 2022, 14:50:51 (UTC-04:00)	1.8 MB	Standard
2021-06-27_00-05.csv	csv	April 18, 2022, 14:50:51 (UTC-04:00)	1.8 MB	Standard
2021-06-27_00-10.csv	csv	April 18, 2022, 14:50:51 (UTC-04:00)	1.8 MB	Standard
2021-06-27_00-15.csv	csv	April 18, 2022, 14:50:51 (UTC-04:00)	1.8 MB	Standard
2021-06-27_00-20.csv	csv	April 18, 2022, 14:50:51 (UTC-04:00)	1.8 MB	Standard
2021-06-27_00-25.csv	csv	April 18, 2022, 14:50:51 (UTC-04:00)	1.8 MB	Standard
2021-06-27_00-30.csv	csv	April 18, 2022, 14:50:51 (UTC-04:00)	1.8 MB	Standard

Lab dataset: Create a table

```
CREATE TABLE mycat.myschema.mytable (
  event_time      TIMESTAMP,
  ip_address      VARCHAR(15),
  app_name        VARCHAR(25),
  process_id      SMALLINT,
  log_type        VARCHAR(15),
  log_level       VARCHAR(15),
  message_id      VARCHAR(15),
  message_details VARCHAR(555)
) WITH (
  external_location = '.../mytable/',
  format = 'TEXTFILE',
  textfile_field_separator = ','
);
```



logdemo

logs_5min_ingest_csv

event_time	timestamp(3)
ip_address	varchar(15)
app_name	varchar(25)
process_id	smallint
log_type	varchar(15)
log_level	varchar(15)
message_id	varchar(15)
message_details	varchar(555)

Lab dataset: Sizing

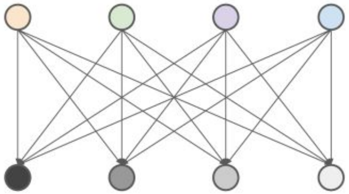
Server logs from June 27, 2021, thru July 31, 2021, generated every 5 minutes

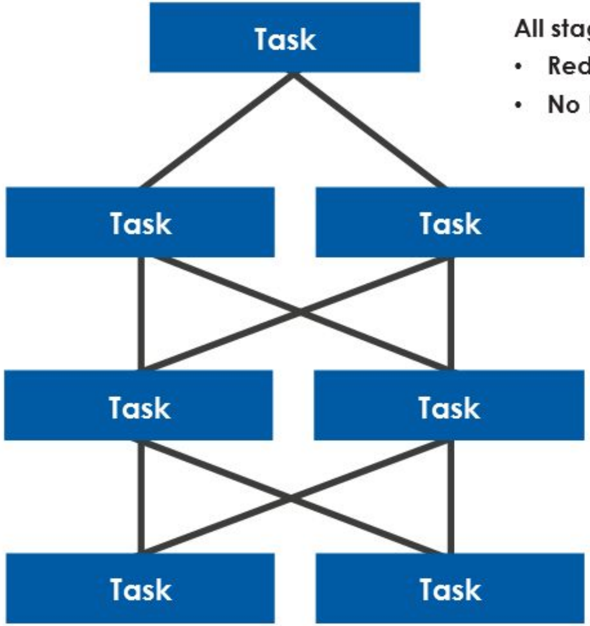
Record size (bytes)	200
Records/ingestion file	10,000
Ingestion file size (bytes)	2,000,000
Files per day	288
Daily file size (bytes)	576,000,000
Records/day rollup	2,880,000
Number of days to load	35
Total number of ingested files	10,080
Total dataset size (bytes)	20,160,000,000
Total number of records	100,800,000

Project & filter early-and-often

Limit the amount of data transferred between query stages

- Operations that work independently on each row are pipelined together into a single task
- Some operations require the entire dataset to be considered and require a new stage (ex: GROUP BY, JOIN, and ORDER BY)
  - Data is shuffled from many:many
- Referencing only columns you need (Projecting) and appropriate WHERE clauses (Filtering) limits
  - Initial data read (especially on columnar files)
  - Amount of data transferred between stages
- These exchanges can get
  - very busy
  - expensive





All stages are pipelined
- Reduced wait time
- No Fault Tolerance

Memory-to-memory Data transfer
- No disc IO
- Data chunk must fit in memory

Lesson summary

Data lake performance: Limiting data exchanges

1. The performance lab exercises will be performed against a server log table.

2. Because clusters are scaled, any improvements in efficiency scale as well.

3. Projection limits the number of columns being queried. Filtering limits the number of rows being queried. Both help to ensure efficiencies across your cluster.

4. Some operations, like GROUP BY, JOIN, and ORDER BY, require data to be exchanged between one stage and another. This is resources intensive..

Data lake performance

File format options

Lesson objectives

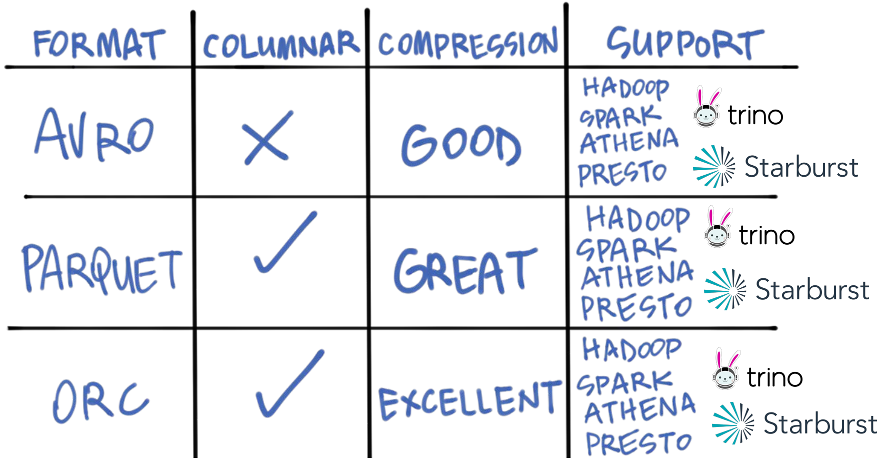
Data lake performance: File format options

1. Explain the difference between row-oriented and columnar file formats.

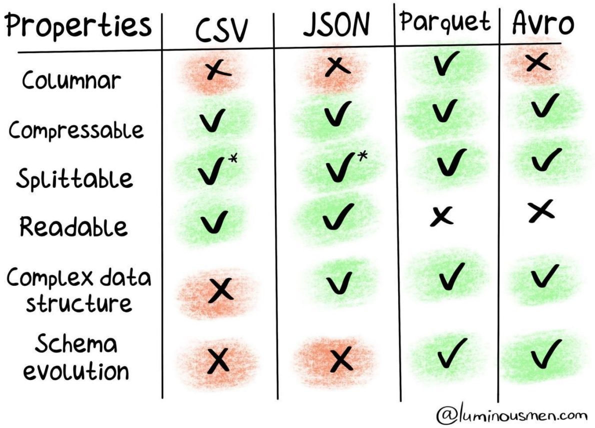
2. Describe the comparative advantages and disadvantages of major file formats.

3. Explain the embedded statistics that are recording inside ORC and Parquet file formats and how they are leveraged.

Multiple file formats exist



FORMAT	COLUMNAR	COMPRESSION	SUPPORT
AVRO	✗	GOOD	HADOOP SPARK ATHENA PRESTO trino Starburst
PARQUET	✓	GREAT	HADOOP SPARK ATHENA PRESTO trino Starburst
ORC	✓	EXCELLENT	HADOOP SPARK ATHENA PRESTO trino Starburst



Properties	CSV	JSON	Parquet	Avro
Columnar	✗	✗	✓	✗
Compressable	✓	✓	✓	✓
Splittable	✓*	✓*	✓	✓
Readable	✓	✓	✗	✗
Complex data structure	✗	✓	✓	✓
Schema evolution	✗	✗	✓	✓

@luminousmen.com

Parquet

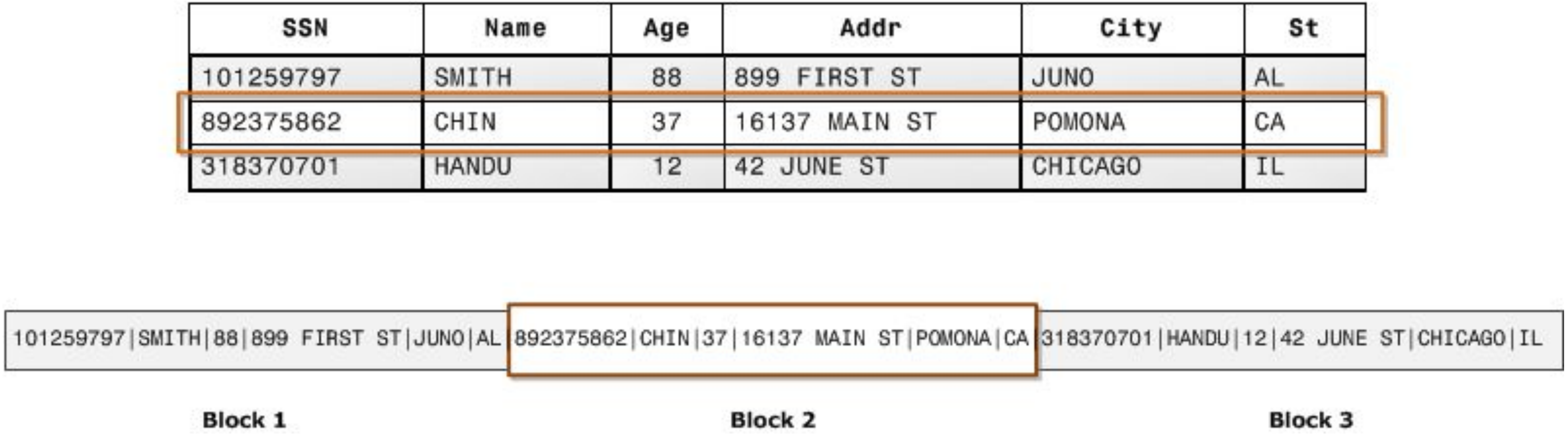
Apache orc™

AVRO™

Row-oriented data storage

All data for a record is kept together and written in order of the fields. Subsequent records are appended to the end of the previous record.

File formats: delimited files (CSV), JSON, Avro



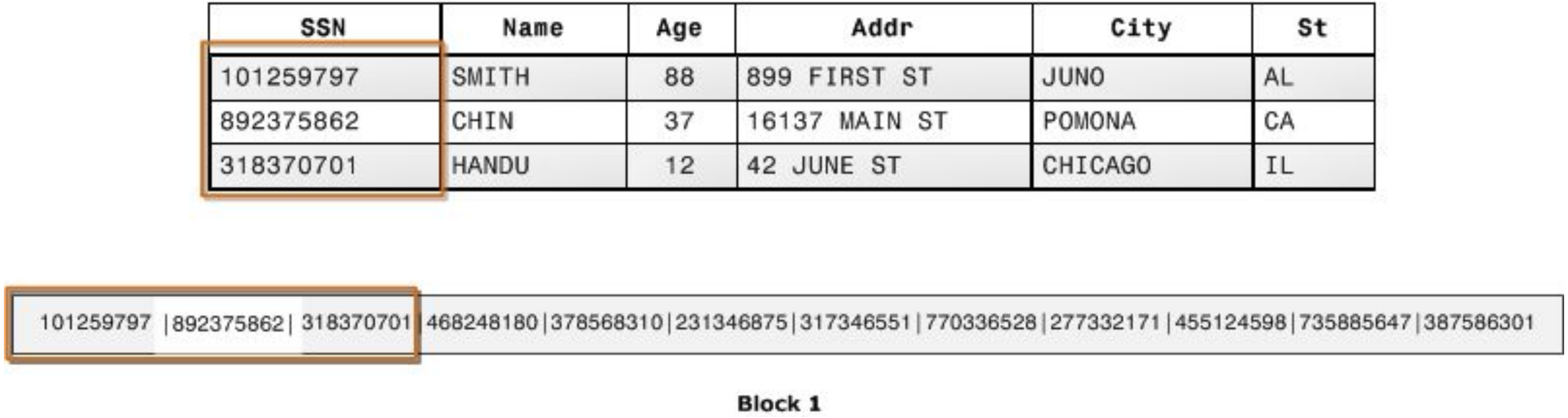
SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL

```
101259797|SMITH|88|899 FIRST ST|JUNO|AL 892375862|CHIN|37|16137 MAIN ST|POMONA|CA 318370701|HANDU|12|42 JUNE ST|CHICAGO|IL
```

Columnar data storage

Data for a specific column (across all records) are stored independently from other columns. Low cardinality fields may employ a dictionary of unique values thereby shrinking the physical size of the data.

File formats: Apache Parquet, Apache ORC

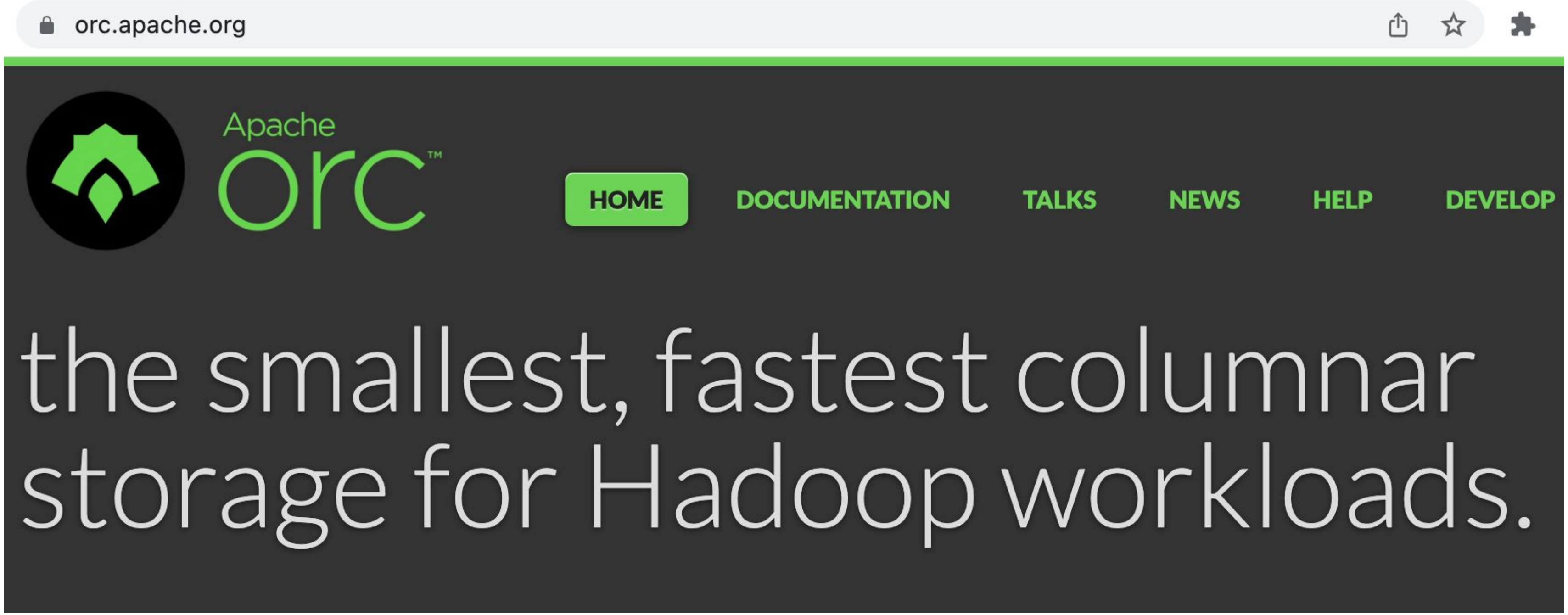


SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL

101259797 \|892375862\| 318370701 \|468248180\|378568310\|231346875\|317346551\|770336528\|277332171\|455124598\|735885647\|387586301

Block 1

Columnar stores excel for analytics



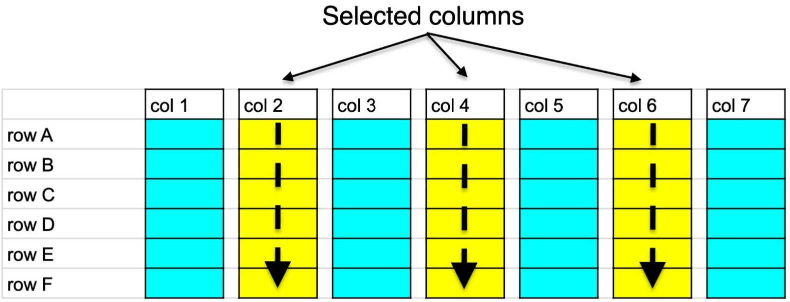
orc.apache.org

the smallest, fastest columnar storage for Hadoop workloads.

[Even Faster ORC (Trino Blog)](#)

Row-oriented vs columnar

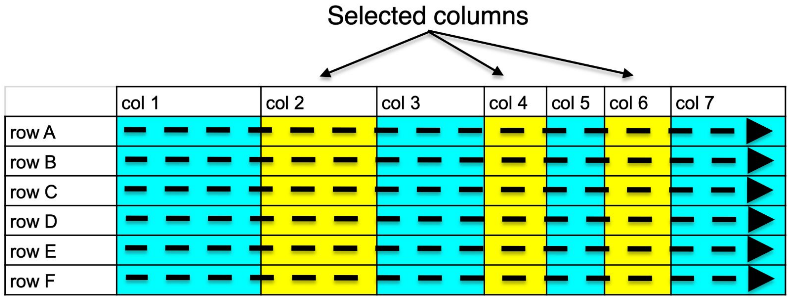
Columnar file formats allow data to be pulled in from only the necessary columns



Columns highly compressed

Read only selected columns

Row-based file formats (like Text and Avro) pull in all data for a given row of data



Rows not compressed

Read the entire row to get selected columns

Instructor demonstration

Exploring file formats (20 mins)

Lesson summary

Data lake performance: File format options

1. Row-oriented file formats organize data by rows. Each new row represents a new record. Examples include CSV, JSON, and AVRO.

2. Columnar file formats organize data by column. Examples include ORC and Parquet.

3. Columnar file formats hold a significant performance advantage, particularly for datasets with low cardinality columns, for queries that retrieve specific columns, and for analytical queries such as aggregations.

4. ORC and Parquet file formats embed additional metadata regarding statistics at the row, stripe/segment, and file levels.

Data lake performance

Small files problem

Starburst

Lesson objectives

Data lake performance: Small File Problem

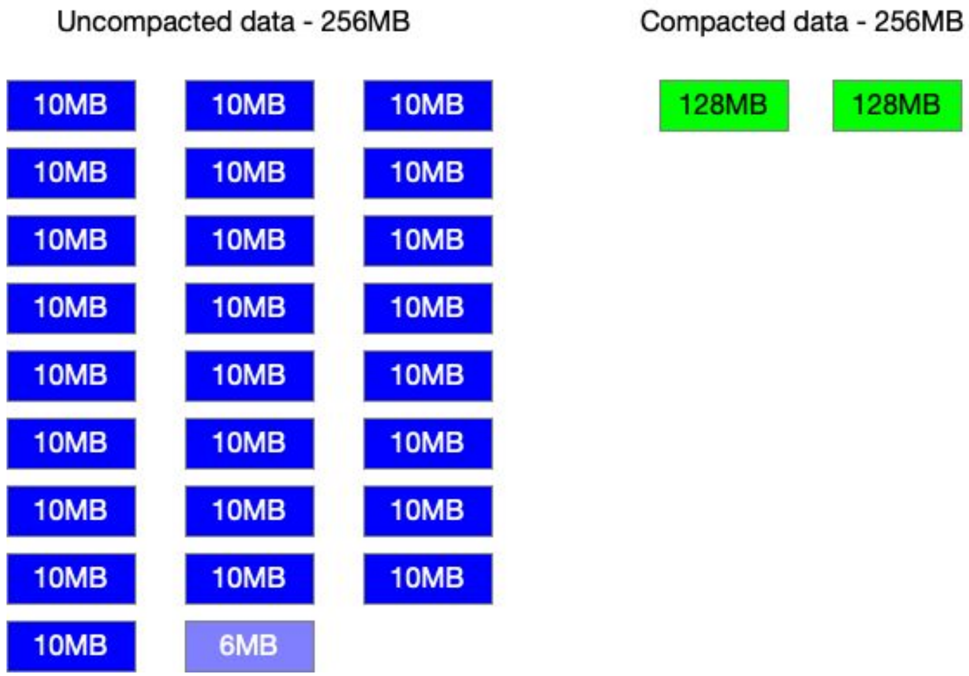
1. Explain the different factors that impact I/O performance costs in a cluster.
2. Describe the nature of the Small File Problem.
3. Demonstrate how file size impact performance in real-world scenarios.

Starburst

File size affects I/O costs

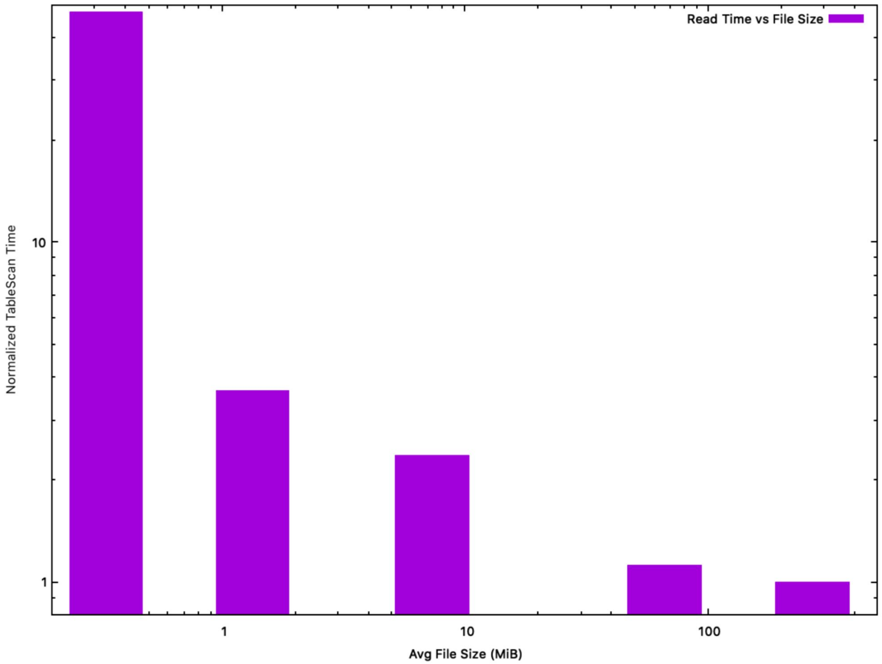
Consider This example

- 26 small files vs 2 large files, assume they contain the same payload.
- In the 26 small files case, lots of files will be opened (relatively expensive I/O overhead)
- In the 2 large files case, only a couple of files will be opened (relatively inexpensive I/O overhead)
- Object stores such as S3 can introduce high latency due to opening and reading many files
- Multiple "compaction" options available



File size recommendations

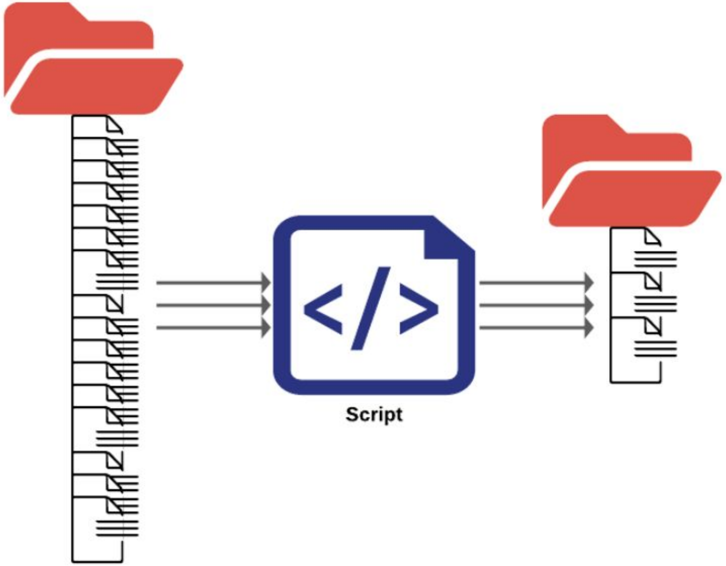
Trino performs best when operating on individual file sizes > 10 MB



- 10 MB is the lower threshold for avoiding the small files problem
- The ideal size range is 64 MB to 512 MB
- The general goal of larger files sizes (>10 MB) is to keep the ratio of data:metadata operations high (i.e. large:small)
- Multiple “compaction” options

Instructor demonstration

File size performance impacts
(10 mins)



Starburst

Hands-on exercises

Lab 1: Create tables with multiple file formats (20 mins)

Lab 2: Using columnar file formats and eliminating small files (40 mins)

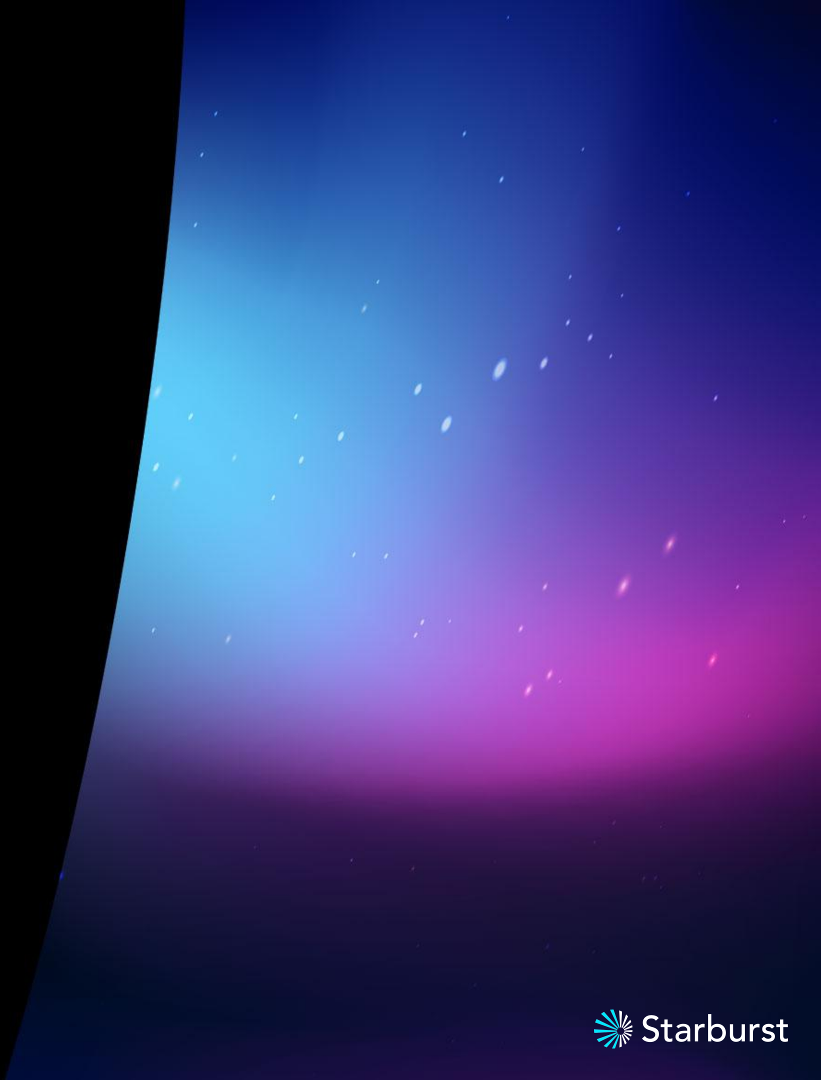
Lesson summary

Data lake performance: Small files problem

1. Smaller files represent a more significant performance overhead when compared to larger ones.

2. Because there are certain unavoidable costs to opening each file, it is not efficient to store data in many small files.

3. Starburst clusters perform best on files larger than 10 MB, and ideally in the 64 MB to 512 MB range.



Data lake performance

Partitioning & bucketing

Starburst

Lesson objectives

Data lake performance: Partitioning & bucketing

1. Explain the role that partitioning plays in data lakes.
2. Explain the role that bucketing plays in data lakes.
3. Explain how sorting and bloom filters provide additional sorting strategies.

Partitioning

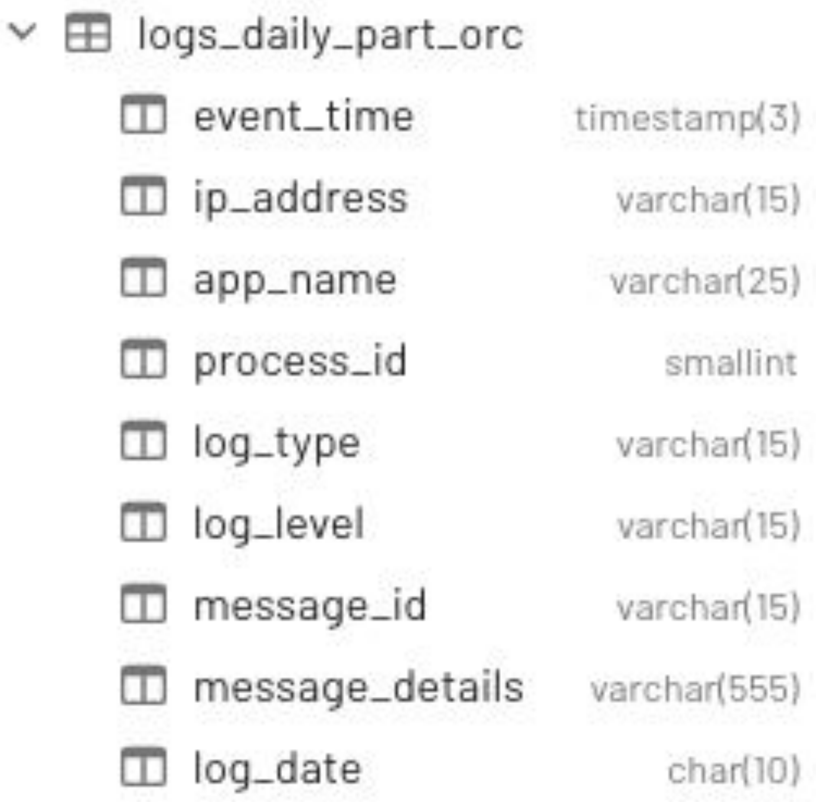
Data lakes employ partitioning as an alternative to indexing

- A partition is just a way to divide data into subdirectories that are designed around logical groupings
- Example: Say you have 5 years of historical data
  - Rather than dump all 5 years into a single folder – you might create a folder for each year, and then put each year's data into the appropriate folder
  - When a query includes the partition key in the predicate or join, the query engine can avoid reading partitions that don't match
- Number of partitions not fixed: Partitions are added as new data is added (IE: New year = new partition)

```
.../cat/sch/hist_tbl
|-- year=2020
|    |-- file1
|    |-- file2
|-- year=2021
|    |-- file3
|    |-- file4
|    |-- file5
|-- year=2022
|    |-- file6
|    |-- file7
```

Define partition column(s) at creation

```
CREATE TABLE mycat.myschema.mytable (
  event_time      TIMESTAMP,
  ip_address      VARCHAR(15),
  app_name        VARCHAR(25),
  process_id      SMALLINT,
  log_type        VARCHAR(15),
  log_level       VARCHAR(15),
  message_id      VARCHAR(15),
  message_details VARCHAR(555),
  log_date        CHAR(10)
) WITH (
  format = 'ORC',
  partitioned_by = ARRAY['log_date']
);
```



logs_daily_part_orc

event_time	timestamp(3)
ip_address	varchar(15)
app_name	varchar(25)
process_id	smallint
log_type	varchar(15)
log_level	varchar(15)
message_id	varchar(15)
message_details	varchar(555)
log_date	char(10)

Instructor demonstration

Partitioning performance impacts (10 mins)

Bucketing

Another technique for dividing data into more manageable chunks is bucketing

- While partitions logical subdivide data into folders, buckets control the number of files within a folder
- With bucketing you choose a fixed number of buckets/files up front
  - Decide bucket count based on desired/ideal file size
- A consistent algorithm is used to determine the bucket assignment
  - Bucketing keeps data that has the same key value in the same bucket
  - This can improve performance as matching data is always stored together in the same bucket

BY ITSELF

```
.../cat/sch/hist_tbl
|-- bucketfile0
|-- bucketfile1
|-- bucketfile2
```

w/PARTITIONING

```
.../cat/sch/hist_tbl
|-- year=2020
|    |-- bucketfile0
|    |-- bucketfile1
|    |-- bucketfile2
|-- year=2021
|    |-- bucketfile0
|    |-- bucketfile1
|    |-- bucketfile2
|-- year=2022
|    |-- bucketfile0
|    |-- bucketfile1
|    |-- bucketfile2
```

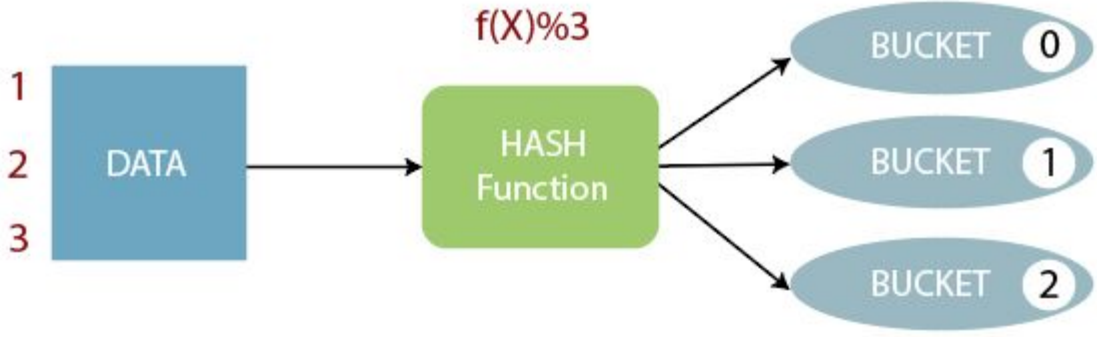
Define bucketing details at creation

```
CREATE TABLE hive.web.page_views (
  view_time timestamp,
  user_id bigint,
  page_url varchar,
  ds date,
  country varchar
)
WITH (
  format = 'ORC',
  partitioned_by = ARRAY['ds', 'country'],
  bucketed_by = ARRAY['user_id'],
  bucket_count = 50
);
```

Hashing - the bucket assignment algorithm

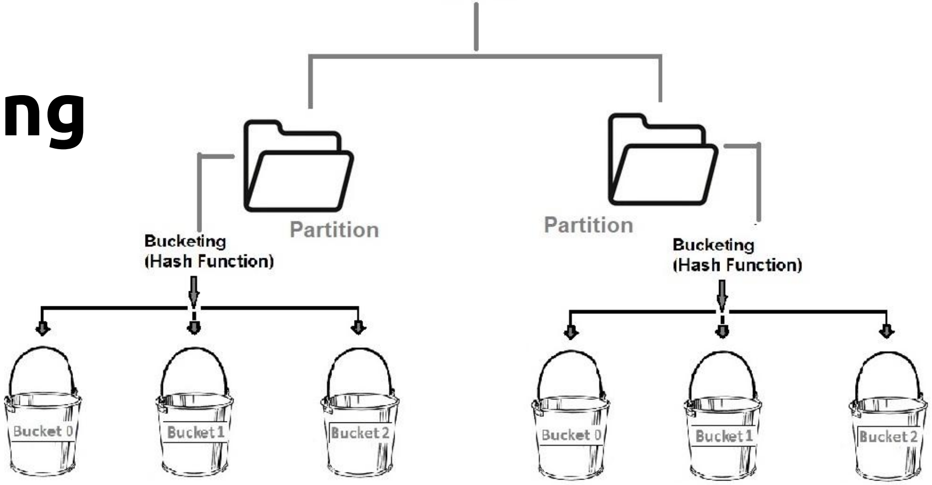
A hash function is way to map data of arbitrary size into a fixed number of buckets

- An example of a simple hash function is “modulo” which provides the “remainder” after dividing two values
- For example: 4 mod 3 = 1 because 4 divided by 3 leaves a remainder of 1
  - Hash functions are used to map values in joins to buckets in a hash table, or to map them to specific server nodes for processing



[Hashing for Trino Table Buckets (YouTube)]

Bucketing vs partitioning



Each solves a different problem

- Both useful to limit the amount of data to read
- Partitions
    - Implemented using folder structures
    - Work best on low cardinality columns that have fairly uniform distribution
- Buckets
    - Implemented by splitting files
    - A way to continue to subdivide data that may not make sense to partition such as high cardinality and/or highly skewed columns
- BE CAREFUL OF CREATING SMALL FILES USING EITHER/BOTH OF THESE FEATURES

Hands-on exercise

Lab 3: Exploring table partitioning and bucketing (40 mins)

Lesson summary

Data lake performance: Partitioning and bucketing

1. Partitioning separates data into folders based on a low cardinality column's value and allows a WHERE clause to determine a subset of these folders to be read.

2. Using a hashing algorithm, bucketing ensures all data with the same value of a high cardinality column is persisted together in the same file thus allowing a WHERE clause to determine which file(s) to read.

3. Total order sorting across files allows the statistics associated with columnar formats to be utilized by the query engine to quickly stop reading a file if a particular value, or range of values, could not be present.

4. A bloom filter is a probabilistic technique that can allow the majority of files to not have to be read when querying for rare values in high cardinality columns.