Building Reliable Data Lake with Spark and Delta Lake

Mohit Batra
Founder, Crystal Talks

linkedin.com/in/mohitbatra

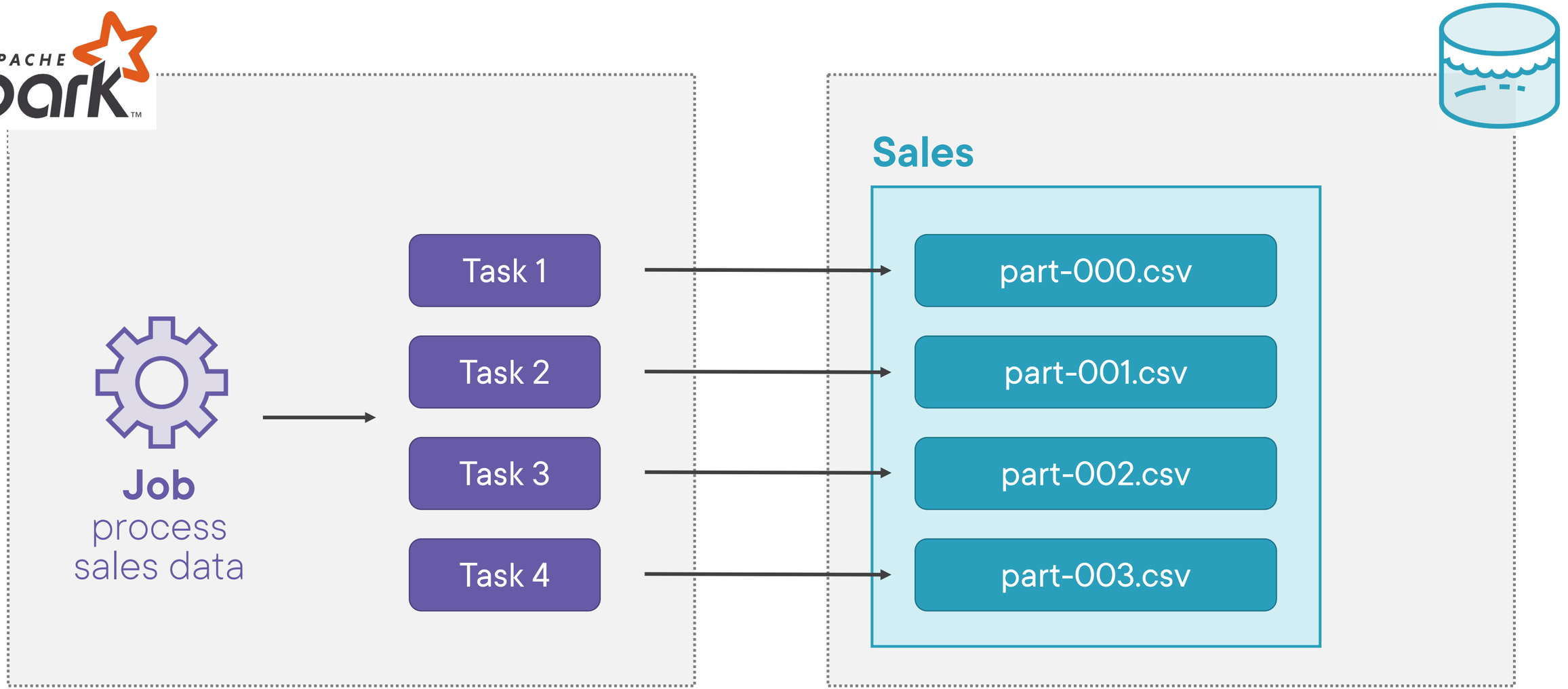
Overview

- Understand the need for Delta Lake
- How Delta Lake works?
- ACID guarantees on Delta Lake
- Create Delta Tables
- Insert data to Delta Table
- Perform DML operations
- Apply table constraints
- Access data using time travel

Need for Delta Lake with Spark

Data Lake is a central repository to store all types of data at any scale, in the form of files

How Spark Writes to Data Lake?



ACID Guarantees on Transactions

Each transaction in a Database provides ACID guarantees

ACID Guarantees on Transactions

Atomicity

All or no changes are written

Id	Name	Salary
1	A	10000
2	B	17000
3	C	12000

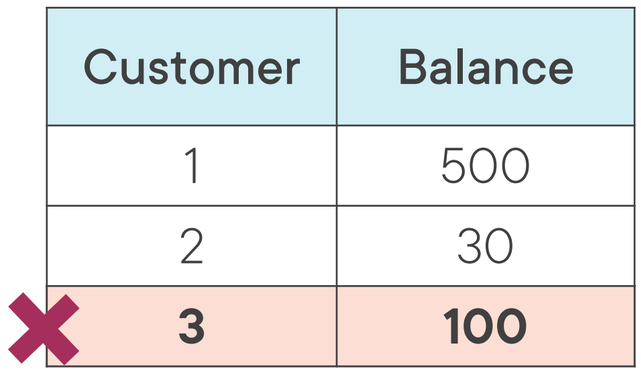
4	D	11000
5	E	14000

ACID Guarantees on Transactions

Consistency

Data always remain
in valid state

Withdraw 200
where Customer = 3



Customer	Balance
1	500
2	30
3	**100**

Constraint
Balance >= 0

ACID Guarantees on Transactions

Isolation

Transaction must run isolated from other processes

Writer

4	D
5	E

Reader

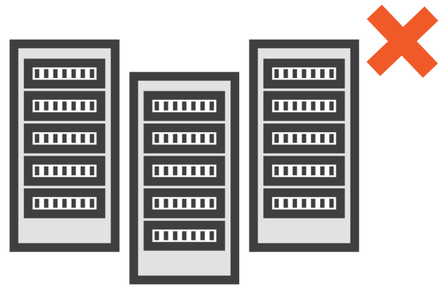
Id	Name
1	A
2	B
3	C

ACID Guarantees on Transactions

Durability

Once committed, data persists even if system fails

Id	Name
1	A
2	B
3	C



Data Lake does NOT provide
ACID guarantees

1 – Job Failure in Appending Data

Writer

Append 2 more part files using 2 tasks

Job failed with runtime error – Part file 4 could not be written

Part File 4 ✖

Part File 1

Part File 2

Part File 3

Reader

Read the folder (reads part files 1, 2 & 3)

Reads inconsistent data

Breaks Atomicity and Consistency!

2 – Job Failure in Overwriting Data

Writer

Overwrite existing =
Delete existing
+ Write 2 new part files

Job failed with runtime error – New part file 2 could not be written

Part File 2 ✖

Reader

Read the folder
(reads part file 1 only)

Reads inconsistent data
+ Previous data is lost

Breaks Atomicity, Consistency and Durability!

3 – Simultaneous Read / Write

Writer

Write 2 more part files using 2 tasks

Only Part file 3 is written. Part 4 is still getting processed

Part File 4

Part File 1

Part File 2

Part File 3

Reader

Read the folder while writing is in progress (reads part files 1, 2 & 3)

Reads inconsistent data

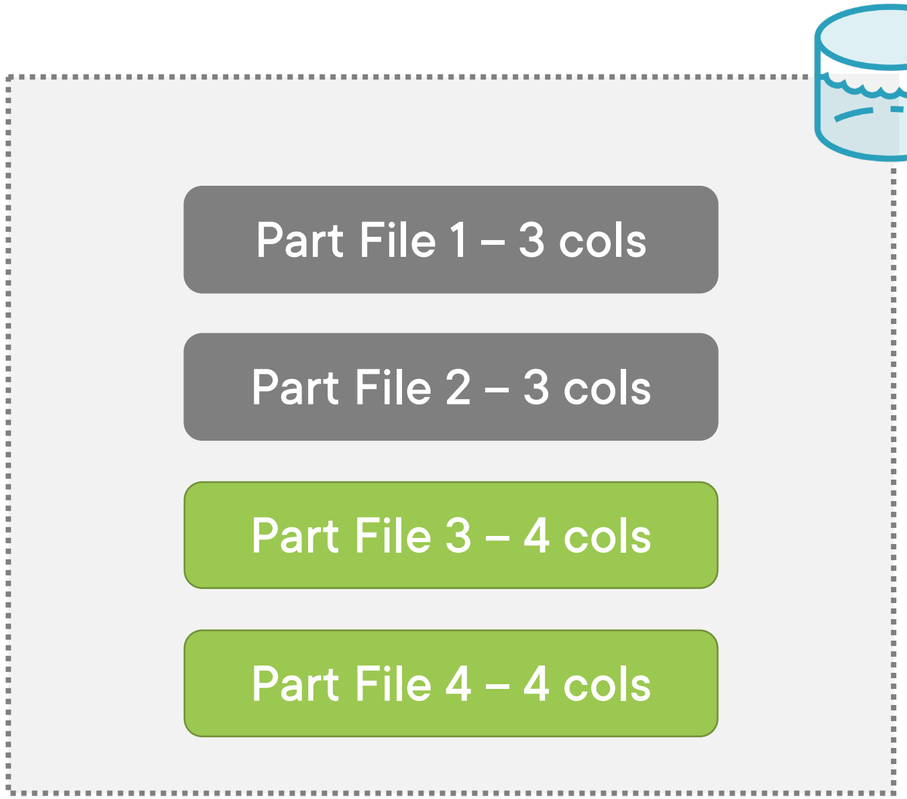
Breaks Consistency and Isolation!

4 – Appending Data with New Schema

Writer

Write 2 more part files with 4 output columns

New output files have 4 columns. No schema validation before writing



Reader

Read the folder (reads all files)

May see 3 or 4 columns – depends on which file is read first

Breaks Consistency!

Challenges with Data Lake

Data reliability issues

- Data corruption because of failures – no rollback!
- No data validation
- Consistency issues while reading data

Handling Batch and Streaming data together is tough

No updates / deletes / merge on files

- Difficult to implement GDPR / CCPA compliance

Data quality issues

- Schema isn't verified before writing
- Cannot apply checks on data

Query performance issues

Difficult to maintain historical versions of data

Delta Lake can help us
solve these challenges!

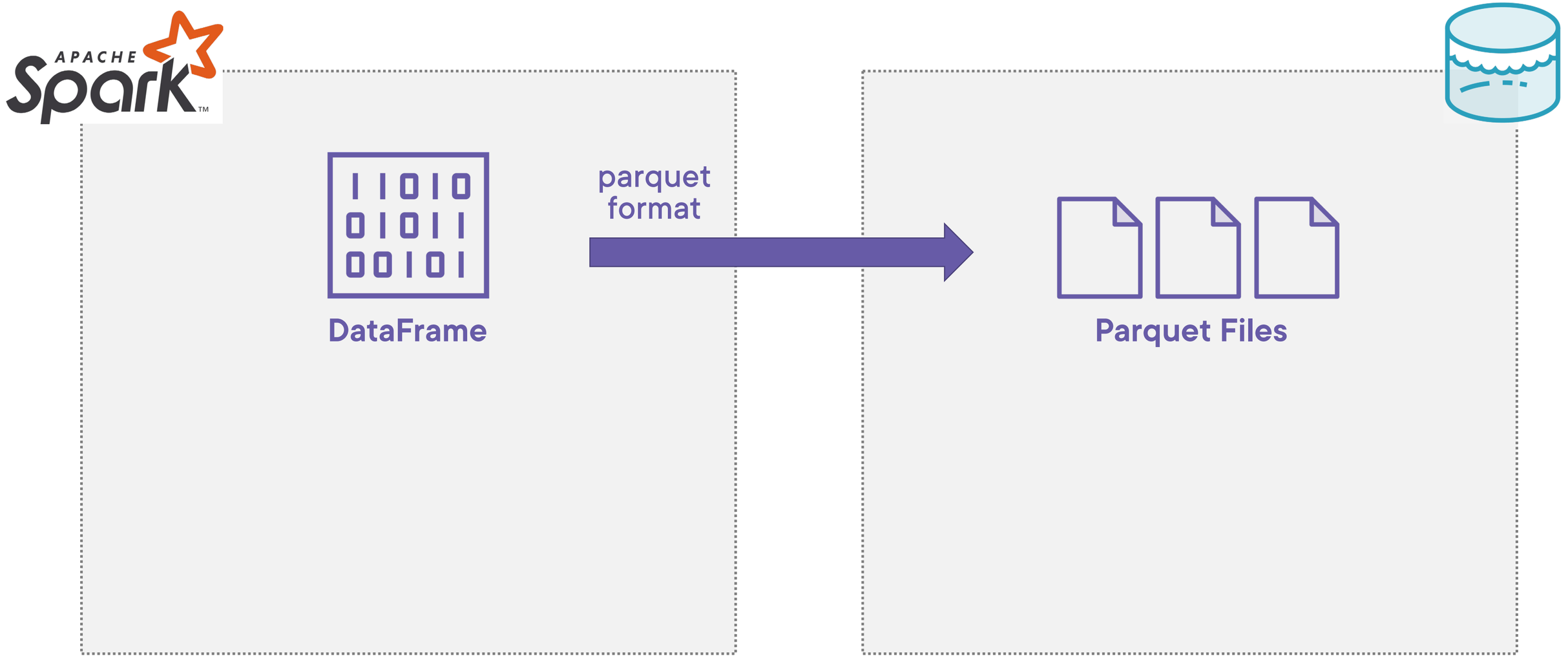
But is Delta Lake the only option?

Apache Iceberg
Apache Hudi

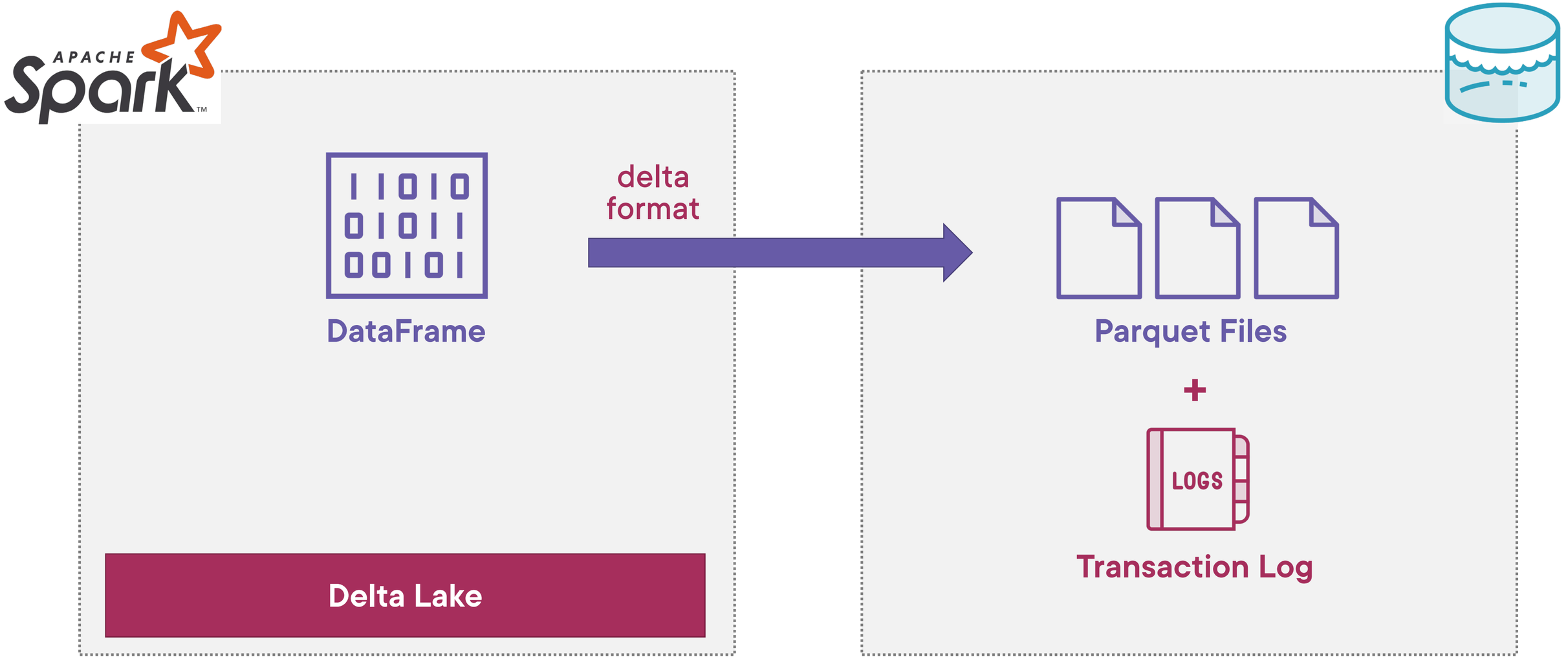
How Delta Lake Works?

Delta Lake is an open-source storage layer that brings reliability to Data Lakes

Writing Data in Parquet Format

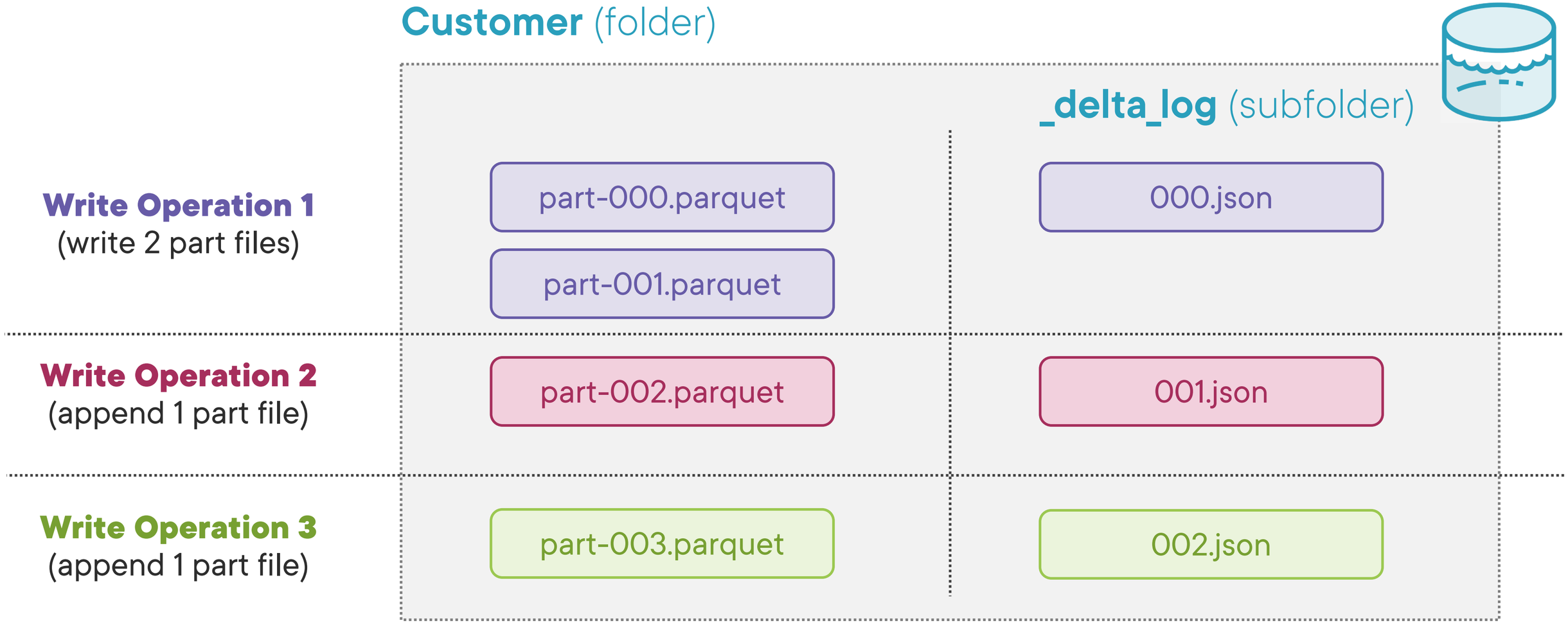


Writing Data in Delta Format



```
(
 dataframe
      .write
      .format("delta")    # like other formats - csv, parquet etc.
      .save(filepath)
)
```

Save DataFrame in Delta Format



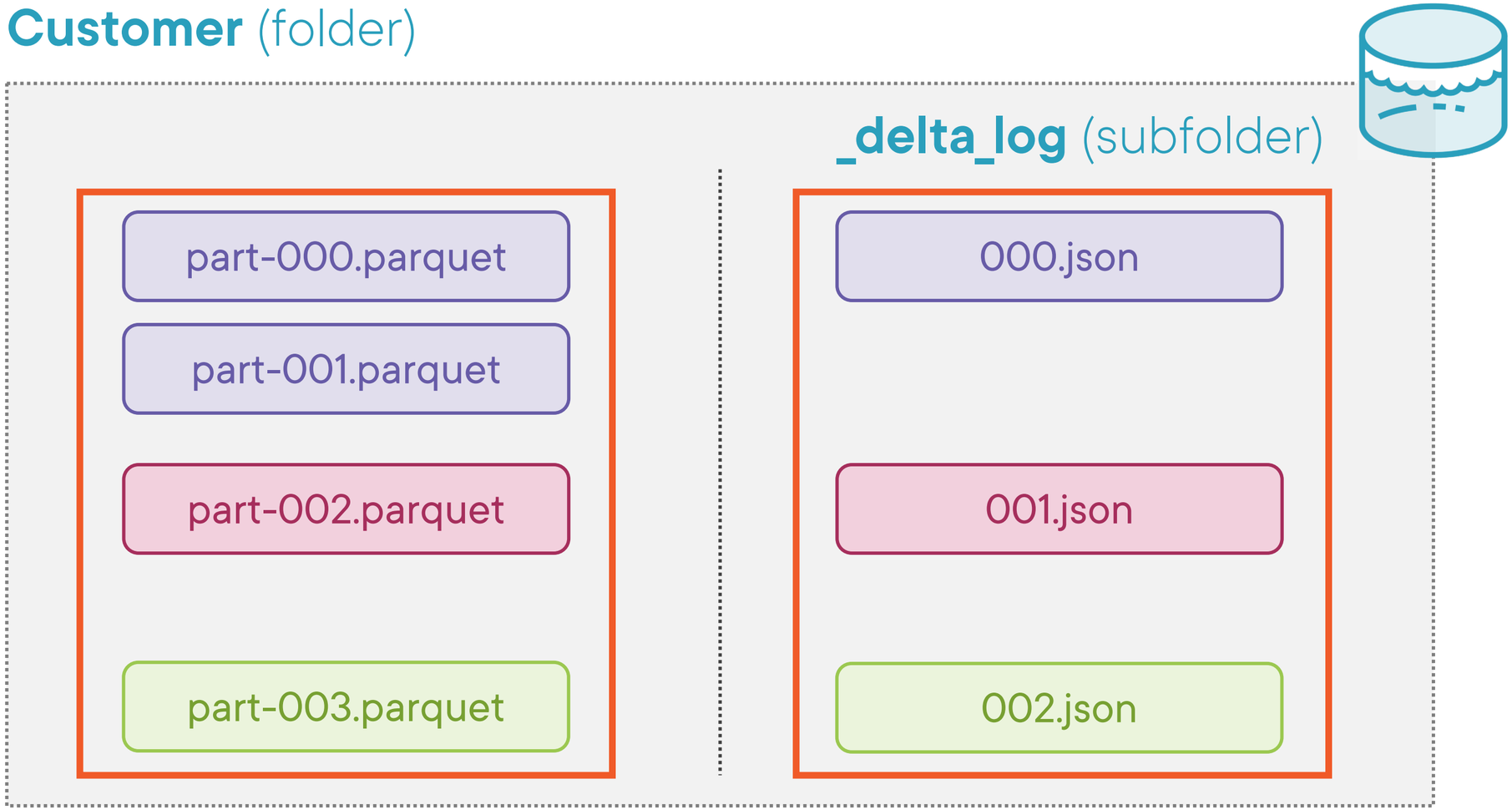
For each write operation:

- Part files are written first
- A transaction log file is added to _delta_log folder in JSON format

Read Operation

Reads log files - 000.json, 001.json, 002.json

Reads 4 part files, based on log information



For each read operation:

- Transaction log files are read first
- Part files are then read based on log files

Customer (folder)

_delta_log (subfolder)

Create Operation
(with 2 part files)

Id	Name
1	A
2	B

Part 1

Id	Name
3	C
4	D

Part 2

Operation	File Name
Add	Part 1
Add	Part 2

000.json

Insert Operation
(append 1 part file)

Id	Name
5	E
6	F

Part 3

Operation	File Name
Add	Part 3

001.json

Update Operation
(Change name from
A to AA where Id = 1)

Id	Name
1	AA
2	B

Part 4

Add modified row

Copy
unchanged rows

Operation	File Name
Remove	Part 1
Add	Part 4

002.json

Customer (folder)

Read all records

~~**Part 1**~~

Part 2

Part 3

Part 4

Id	Name
1	A
2	B

Part 1

Id	Name
3	C
4	D

Part 2

Id	Name
5	E
6	F

Part 3

Id	Name
1	AA
2	B

Part 4

_delta_log (subfolder)

Operation	File Name
Add	Part 1
Add	Part 2

000.json

Operation	File Name
Add	Part 3

001.json

Operation	File Name
Remove	Part 1
Add	Part 4

002.json

Customer (folder)

Read all records

Id	Name
3	C
4	D
5	E
6	F
1	AA
2	B

Id	Name
1	A
2	B

Part 1

Id	Name
3	C
4	D

Part 2

Id	Name
5	E
6	F

Part 3

Id	Name
1	AA
2	B

Part 4

_delta_log (subfolder)

Operation	File Name
Add	Part 1
Add	Part 2

000.json

Operation	File Name
Add	Part 3

001.json

Operation	File Name
Remove	Part 1
Add	Part 4

002.json

Delta Lake Features

Provides ACID Guarantees

- No data corruption because of failures
- Data consistency while reading data

Perform inserts / updates / deletes

Schema enforcement and evolution

Protect data using Time Travel

Handle batch & streaming data together

Apply data quality checks

Performance improvements using statistics

...and much more

ACID Guarantees on Delta Lake

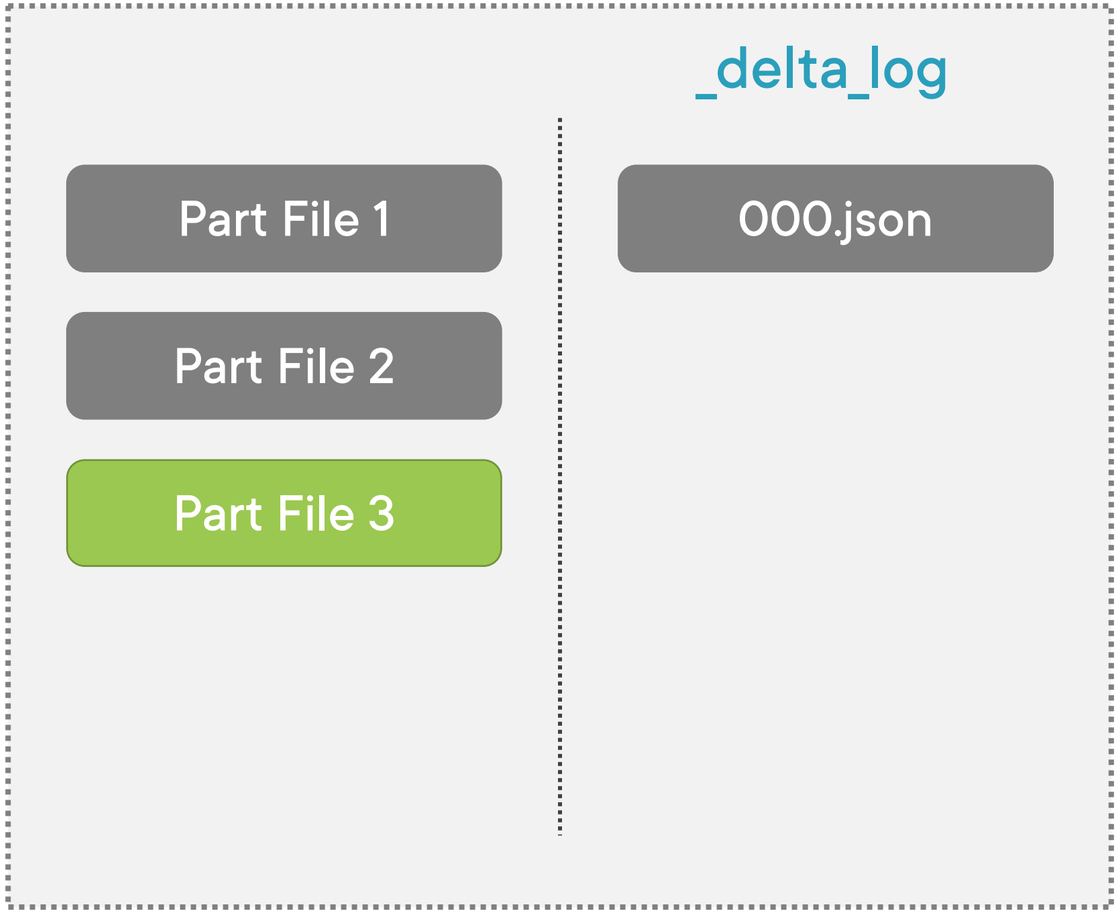
1 – Job Failure in Appending Data

Writer

Append 2 more part files using 2 tasks

Job failed with runtime error – Part file 4 could not be written

Part File 4 ✖



Reader

Reads log file (000.json)

Only reads Part files 1 & 2

Reads consistent data

2 – Job Failure in Overwriting Data

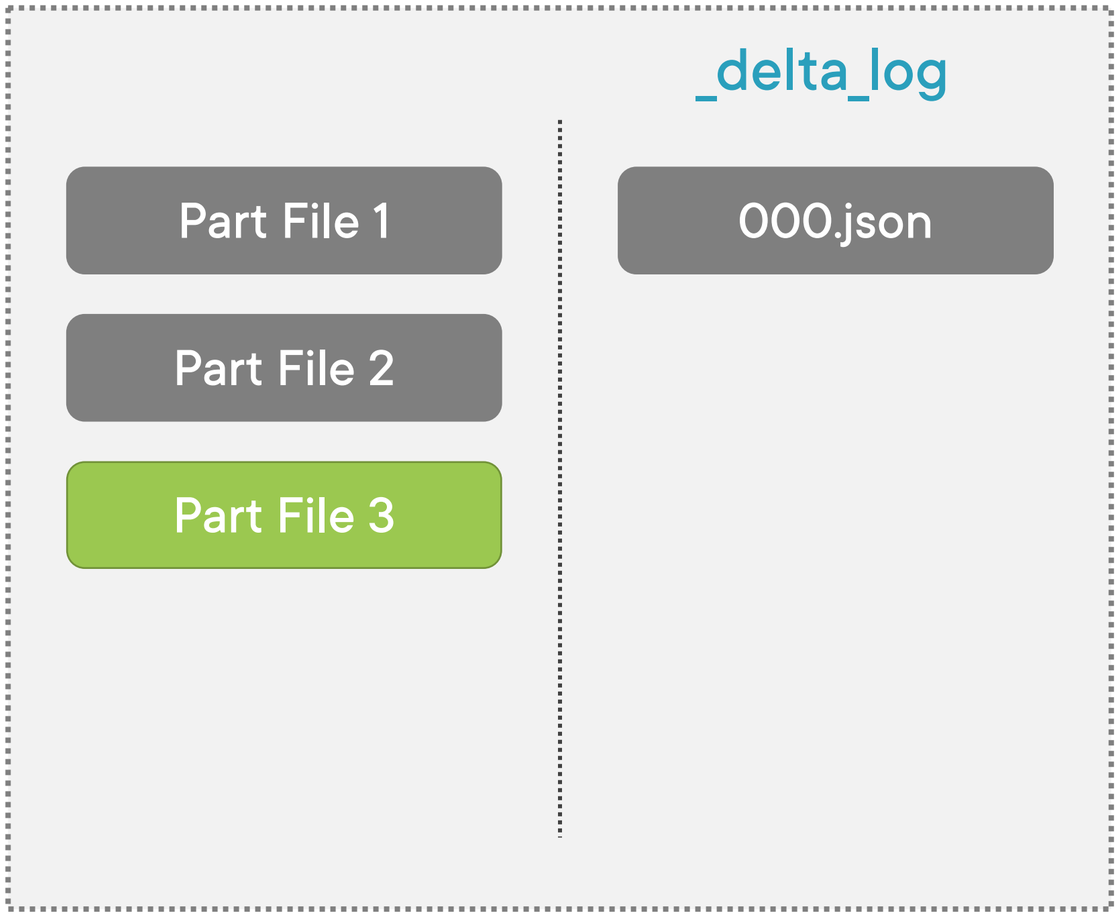
Writer

Overwrite existing data by writing 2 new part files

Existing files are not deleted!

Job failed with runtime error – New part file 4 could not be written

Part File 4 ✖



Reader

Reads log file (000.json)

Only reads Part files 1 & 2

Reads consistent data
+ Previous data is not lost

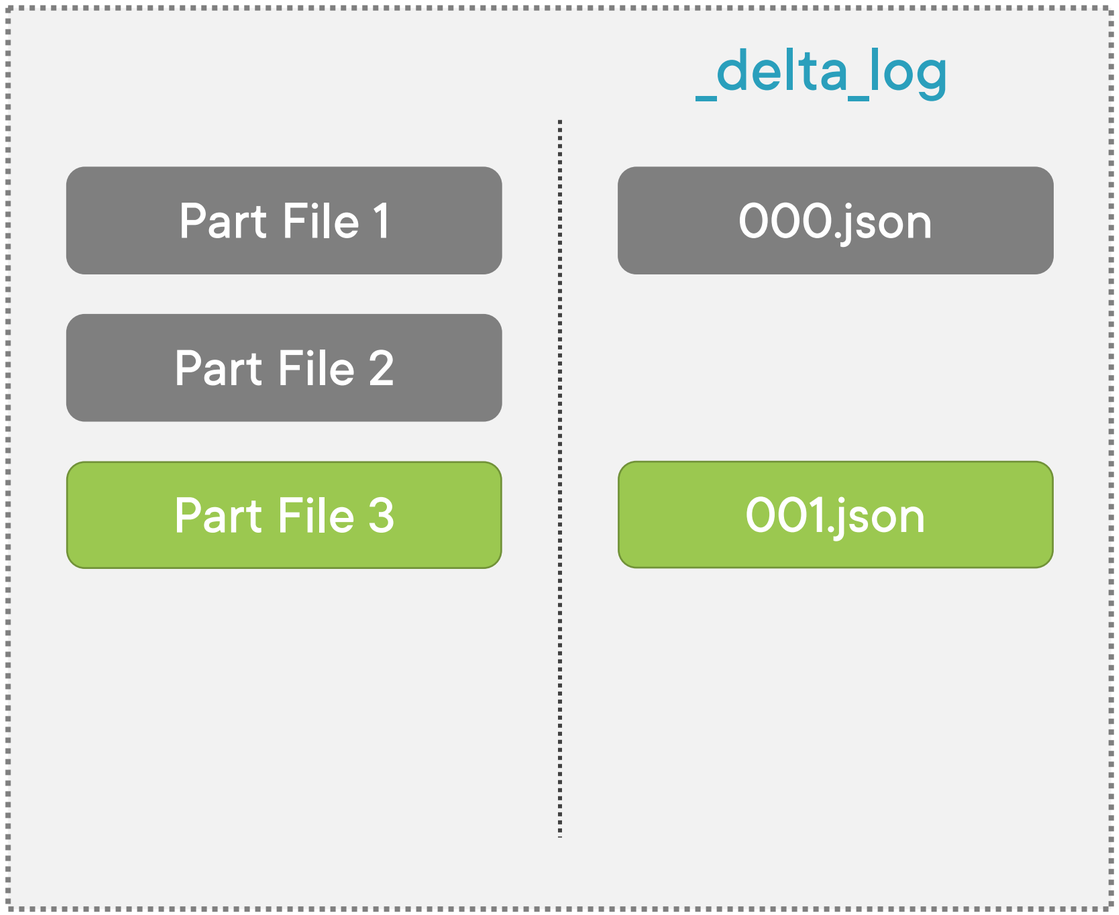
3 – Simultaneous Read / Write

Writer

Write 2 more part files using 2 tasks

Only Part file 3 is written. Part 4 is still getting processed

Part File 4



Reader

Reads log file while writing is in progress (reads 000.json)

Only reads Part files 1 & 2

No dirty reads while writing is in progress

Creating Delta Tables

Sample Transaction Log Entry

```
{"commitInfo":{"timestamp":1640067393982,"userId":"1472626815582251",
    "operation":"WRITE","operationParameters":{"mode":“Overwrite"},
    "operationMetrics":{"numFiles":"2","numOutputRows":“200","numOutputBytes":“14629"}}}
```

```
{"metaData":{"schemaString":"{
   \"fields\":[
              {\"name\":\"Day\",    \"type\":\"integer\", \"nullable\":true, \"metadata\":{}},
              {\"name\":\"RideId\", \"type\":\"integer\", \"nullable\":true, \"metadata\":{}},
              {\"name\":\"Amount\", \"type\":\"double\",  \"nullable\":true, \"metadata\":{}}
            ]
}"}}
```

```
{"add":{"path":"part-000.parquet","size":7284 }}
{"add":{"path":"part-001.parquet","size":7345 }}
```

Sample Transaction Log Entry


```
{"commitInfo":{"timestamp":1640067393982,"userId":"1472626815582251",
    "operation":"WRITE","operationParameters":{"mode":"Overwrite"},
    "operationMetrics":{"numFiles":"2","numOutputRows":“200","numOutputBytes":“14629"}}}
```

```
{"metaData":{"schemaString":"{
   \"fields\":[
              {\"name\":\"Day\",    \"type\":\"integer\", \"nullable\":true, \"metadata\":{}},
              {\"name\":\"RideId\", \"type\":\"integer\", \"nullable\":true, \"metadata\":{}},
              {\"name\":\"Amount\", \"type\":\"double\",  \"nullable\":true, \"metadata\":{}}
            ]
}"}}
```

```
{"add":{"path":"part-000.parquet","size":7284 }}
{"add":{"path":"part-001.parquet","size":7345 }}
```

Sample Transaction Log Entry

```
{"commitInfo":{"timestamp":1640067393982,"userId":"1472626815582251",
    "operation":"WRITE","operationParameters":{"mode":"Overwrite"},
    "operationMetrics":{"numFiles":"2","numOutputRows":“200","numOutputBytes":“14629"}}}
```

```
{"metaData":{"schemaString":"{
   \"fields\":[
              {\"name\":\"Day\",    \"type\":\"integer\", \"nullable\":true, \"metadata\":{}},
              {\"name\":\"RideId\", \"type\":\"integer\", \"nullable\":true, \"metadata\":{}},
              {\"name\":\"Amount\", \"type\":\"double\",  \"nullable\":true, \"metadata\":{}}
            ]
}"}}
```

```
{"add":{"path":"part-000.parquet","size":7284 }}
{"add":{"path":"part-001.parquet","size":7345 }}
```

Sample Transaction Log Entry

```
{"commitInfo":{"timestamp":1640067393982,"userId":"1472626815582251",
    "operation":"WRITE","operationParameters":{"mode":"Append"},
    "operationMetrics":{"numFiles":"2","numOutputBytes":“14629","numOutputRows":"200"}}}
```

```
{"metaData":{"schemaString":"{
   \"fields\":[
              {\"name\":\"Day\",    \"type\":\"integer\", \"nullable\":true, \"metadata\":{}},
              {\"name\":\"RideId\", \"type\":\"integer\", \"nullable\":true, \"metadata\":{}},
              {\"name\":\"Amount\", \"type\":\"double\",  \"nullable\":true, \"metadata\":{}}
            ]
}"}}
```

```
{"add":{"path":"part-000.parquet","size":7284 }}
{"add":{"path":"part-001.parquet","size":7345 }}
```

Inserting Data to Delta Table

Options to Insert Data

INSERT Command
(SQL)

Append DataFrame
(PySpark / Scala)

Performing DML Operations

Applying Table Constraints

Table Constraints

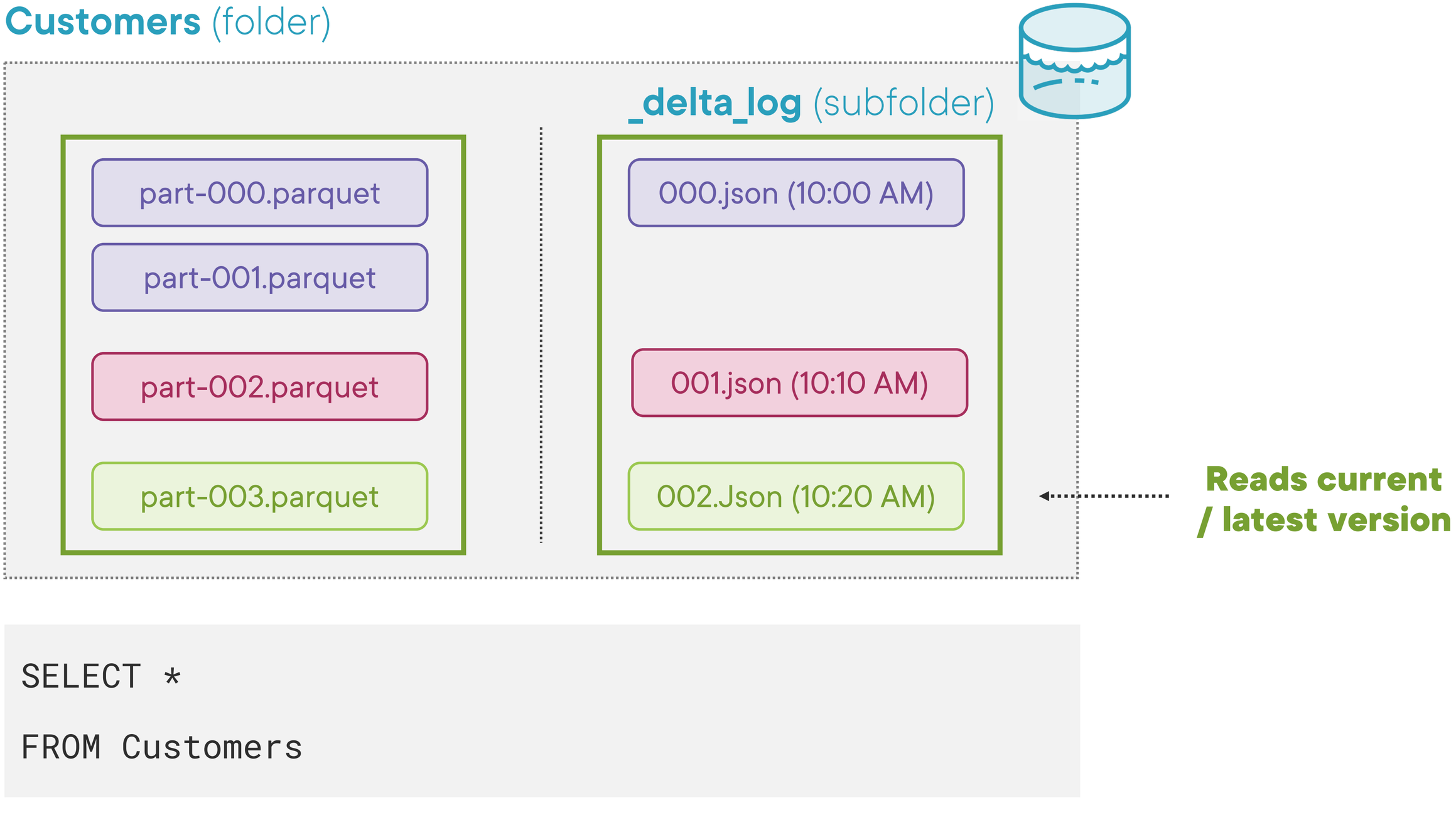
NOT NULL

To prevent column from having NULL values

CHECK

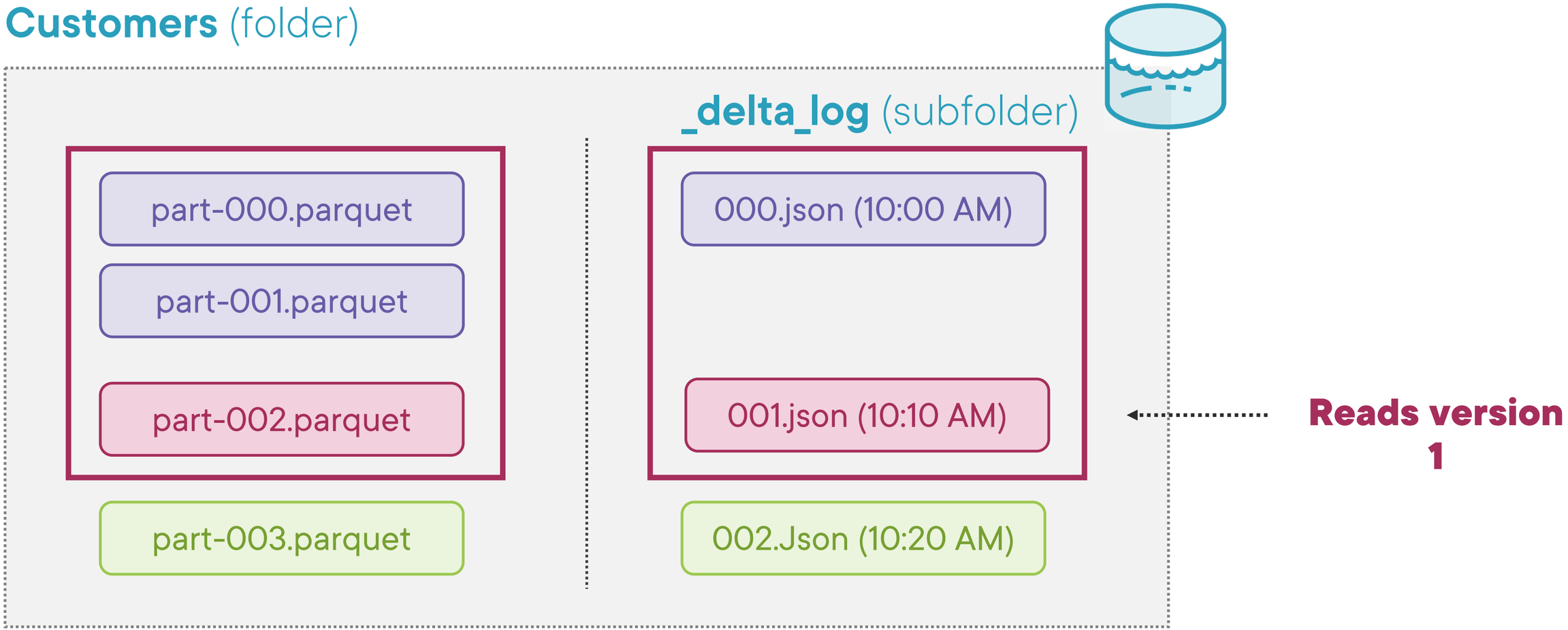
Define conditions to enforce on data in the table

Accessing Data with Time Travel



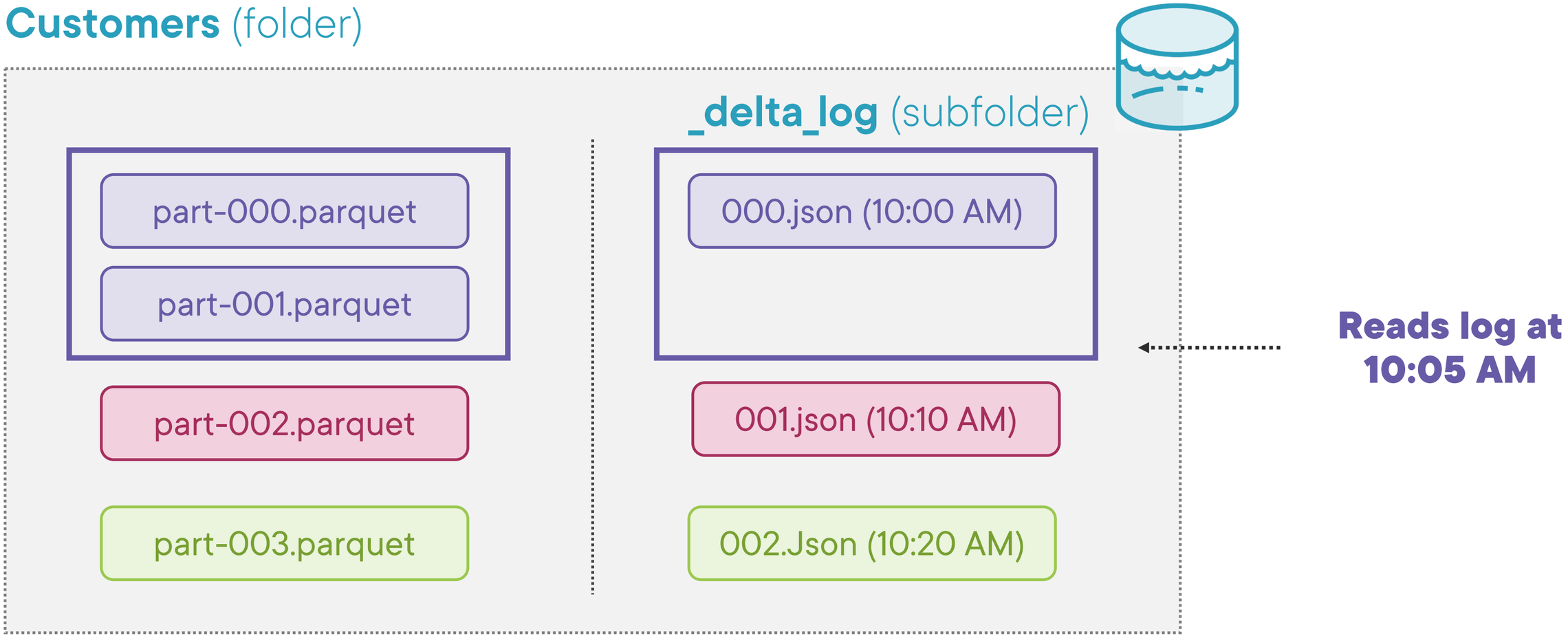

```
SELECT *

FROM Customers
```




```
SELECT *

FROM Customers VERSION AS OF 1
```




```
SELECT *

FROM Customers TIMESTAMP AS OF ‘2022-03-01 10:05’
```

Time Travel allows to access/restore previous snapshot of data, even if data has been modified or deleted

Summary

Delta Lake is storage layer bringing reliability to Lakes

- Stores data in parquet format + Transaction log
- Provides ACID guarantees

Operations

- Write operation: First write the files, then the log
- Read operation: First read the log, then the files

Create Delta Table by storing data in delta format

Various options to add data to Delta Table

- Append DataFrame (Python) & Insert (SQL)

Perform DML operations - Update, Delete & Merge

Apply table constraints – Not Null & Check

Access data using time travel

- Query using version or timestamp, or restore table

Up Next:
Handling Streaming Data with Spark Structured Streaming