

## DP-203: 05- Storage account access tiers

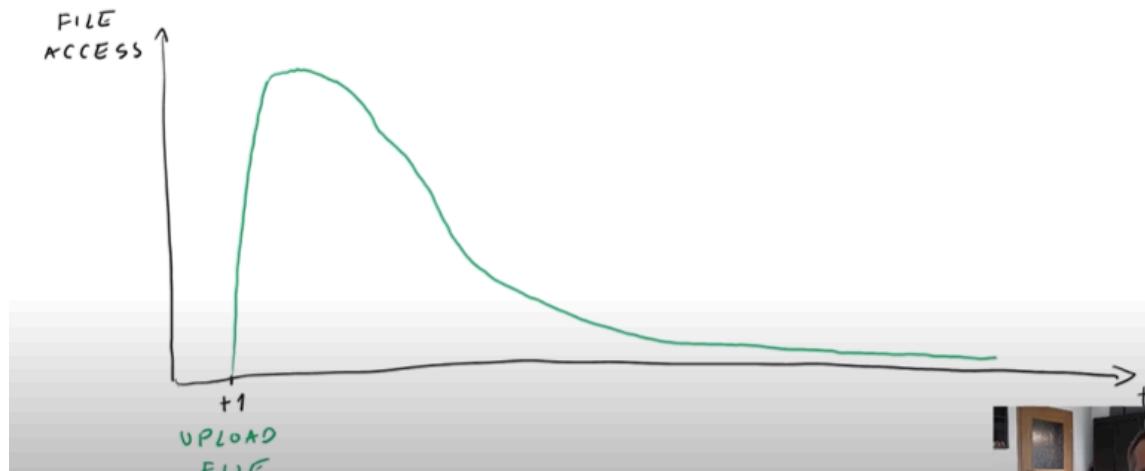
Data usage over time

1. Basically in ADL we pay for two things



+other things such as redundancy factors

2. Let us take a graph ...on Y-axis we have file access(Read/Write operation) and on X-axis we have time



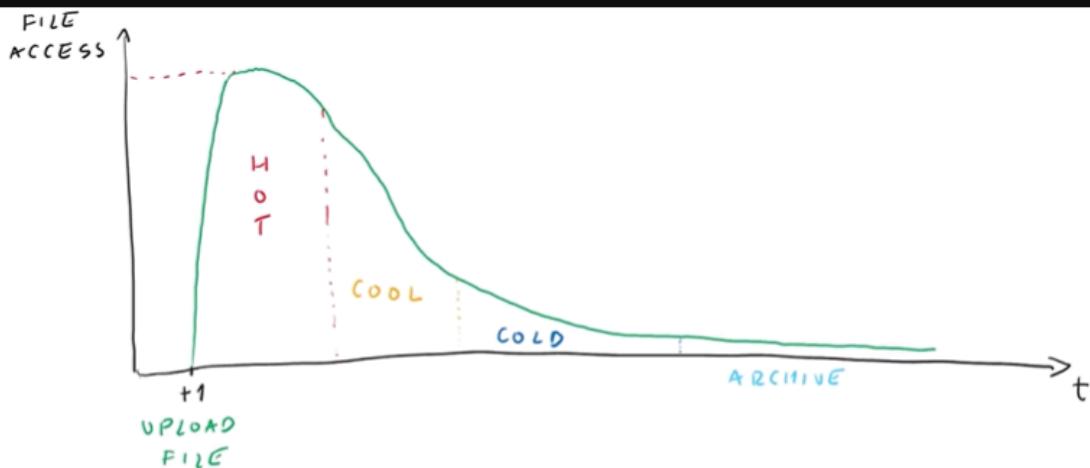
3. Here if we upload a file at time t1(then for 1 week there will be heavy Read/Write Operations) after that file access operation slows down with time
4. So with the help of access tiers we can control our cost spending

## Access tiers

1. Here the access tiers are based on access speed(How fast we can get data), storage costs, access costs(Read/Write operations) and retention period
2. Based on those factors we have

ACCESS TIER	ACCESS SPEED	STORAGE COSTS	ACCESS COSTS	RETENTION PERIOD
HOT	ms	\$\$\$\$\$	\$	
COOL	ms	\$\$\$\$	\$\$	
COLD	ms	\$\$	\$\$\$	
ARCHIVE	hours	\$	\$\$\$\$	

3. Hot access tier is more expensive in storage costs and least expensive in access costs..we can see the pic and understands access tiers
4. When to use which access tier?



5. Like when we upload some files initially..then there will high Read/Write operations..so we use Hot...and later if the data is of no use..then we move it to the other access tier

6. Here we use archive tier to store any legal data etc

TIER	SPEED	COSTS	COSTS
ONLINE	HOT ms	\$\$\$\$\$	\$
	COOL ms	\$\$\$\$	\$\$
	COLD ms	\$\$	\$\$\$
OFFLINE ARCHIVE	hours	\$	\$\$\$\$

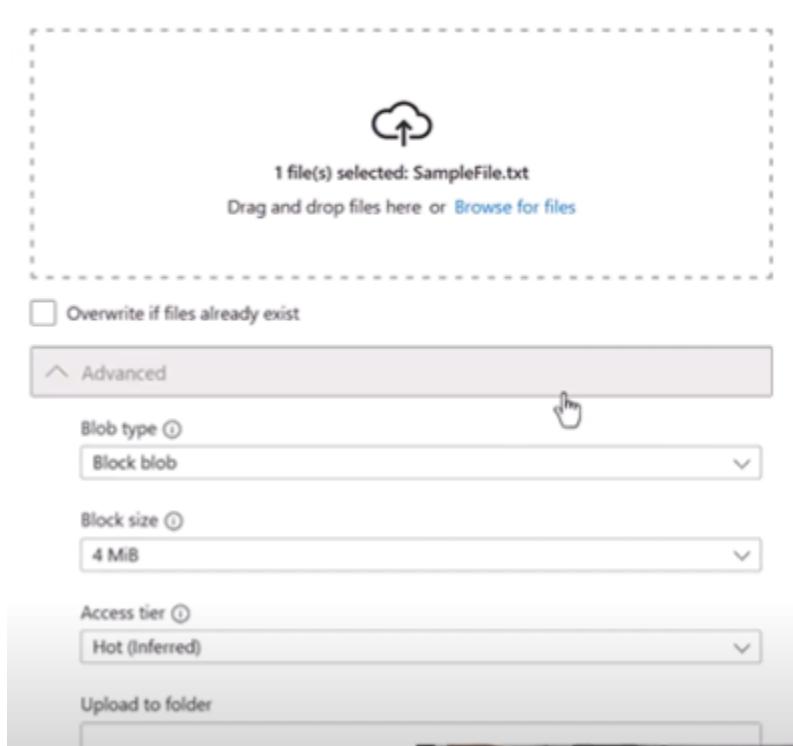
7. To use the data which is in archive tier...we need to move it in any other tier(which takes hours)

## Access tier demo

1. Here first we will create a container ..and inside that we upload a file

The screenshot shows the Azure Storage Blob upload interface. On the left, there's a list of blobs in a container named 'container1'. On the right, a modal window titled 'Upload blob' is open, showing a file named 'SampleFile.txt' selected for upload. The modal includes options for overwriting existing files and advanced settings. At the bottom right of the interface, there's a video feed of a person.

In the advanced section..we can specify the access tier



+ Hot (Inferred)

Hot

Cool

Cold

Archive

2. Hot tier will be default..if we do not select any tier

3. We can also use change tier option..to move files bw the tiers

The screenshot shows the Azure Storage Blob container overview page. A 'Change tier' dialog is open on the right side. The dialog title is 'Change tier' and it says 'Samplefile.txt'. Below the title, there is a message: 'Optimize storage costs by placing your data in the appropriate access tier. Learn more.' A dropdown menu labeled 'Access tier' is set to 'Hot'. Below the dropdown, it says 'Access tier last modified 9/3/2023, 9:36:27 AM'. On the left, the main container view shows a table with columns: Name, Modified, Access tier, Archive status, and Blob. There is one item listed: 'SampleFile.txt' with a modified date of '9/3/2023, 9:36:27 AM', access tier 'Cool', archive status 'None', and blob type 'Blob'.

4. If we move for file to archive tier..it gives us

## Access tier

### Archive

**i** Setting the access tier to "Archive" will make your blob inaccessible until it is rehydrated back to "Hot" or "Cool", which may take several hours.

## Access tier last modified

9/3/2023, 9:37:46 AM

5. And when we try to access the file in archive tier then we get

The screenshot shows the Azure Storage Blob container overview page. A large error message 'You do not have access' is displayed in the center. To the right of the message is a cloud icon with a slash through it, indicating access is denied. Below the message, it says 'This request is not authorized to perform this operation using this permission.' A detailed error summary is shown in a table:

Session ID	72b85dc8a40d4084956584d6beaf5a4ee
Extension	Microsoft_Azure_Storage
Error code	403
Resource ID	/subscriptions/1c461dd7-4fb5-4d22-b200-e052f75b9c...
Content	BlobPropertiesBladeV2
Storage Request ID	809adef4-101e-0070-503a-def28000000

Below the summary table, under 'Details', there is a note: 'This request is not authorized to perform this operation using this permission. RequestId:809adef4-101e-0070-503a-def28000000 Time:2023-09-03T07:41:32.8956996Z'.

6. To get access to this file...first we have to rehydrate it into other access tiers

The screenshot shows the Azure Storage Explorer interface. At the top, there's a breadcrumb navigation: '53725416796 | Overview > dp203datalaketybuldemo | Containers >'. On the right, a message says 'Successfully updated access tier' and 'Successfully updated access tier for blob 'SampleFile.txt''. Below the header, there are several buttons: Upload, Add Directory, Refresh, Rename, Delete, Change tier, Acquire lease, Break lease, and Give feedback. Underneath these are sections for 'Authentication method: Access key (Switch to Azure AD User Account)' and 'Location: container1'. A search bar with placeholder 'Search blobs by prefix (case-sensitive)' and a 'Show deleted objects' checkbox are also present. The main table lists blobs with columns: Name, Modified, Access tier, Archive status, Blob type, Size, Lease state, and three ellipsis buttons. The 'Archive status' column for 'SampleFile.txt' is highlighted with a red border.

7.

## Datalake Pricing

### Data storage

	Premium	Hot	Cool	Cold	Archive
First 50 terabyte (TB) / month	\$0.15 per GB	\$0.0184 per GB	\$0.01 per GB	\$0.0036 per GB	\$0.00099 per GB
Next 450 TB / month	\$0.15 per GB	\$0.0177 per GB	\$0.01 per GB	\$0.0036 per GB	\$0.00099 per GB
Over 500 TB / month	\$0.15 per GB	\$0.0169 per GB	\$0.01 per GB	\$0.0036 per GB	\$0.00099 per GB

1.

here we can see to store the data HOT is expensive and archive is cheap

2. Transaction pricing(Reads/Writes)

### Transaction

	Premium	Hot	Cool	Cold	Archive
Write Operations* (every 4MB, per 10,000)	\$0.0228	\$0.065	\$0.13	\$0.234	\$0.13
Read Operations** (every 4MB, per 10,000)	\$0.00182	\$0.0052	\$0.013	\$0.13	\$6.50
Query Acceleration - Data Scanned (per GB)	N/A	\$0.002	\$0.002	\$0.002	N/A
Query Acceleration - Data Returned (per GB)	N/A	\$0.0007	\$0.01	\$0.01	N/A

\*The following API calls are considered write operations: AppendFile, CreateFilesystem, CreatePath, CreatePathFile, FlushFile, SetFileProperties, SetFilesystemProperties, RenameFile, RenamePathFile, CopyFile  
\*\*The following API calls are considered read operations: ReadFile, ListFilesystemFile

## Retention Period

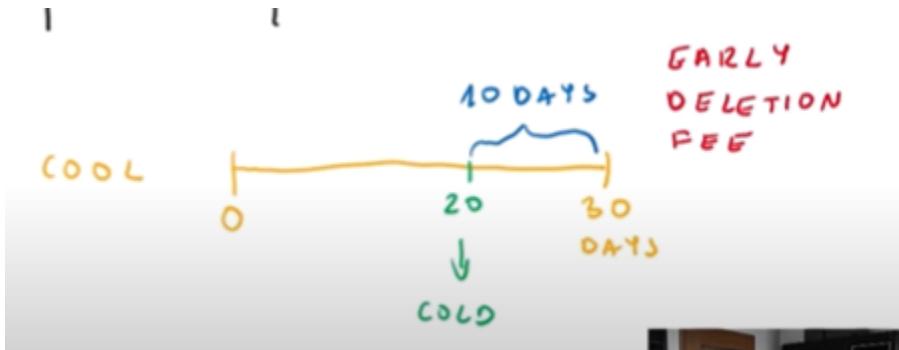
- It is nothing but for how long will the file stay in one tier
- So here the file has to be present in the cool,cold and archive for specified retention period

ACCESS TIER	ACCESS SPEED	STORAGE COSTS	ACCESS COSTS	RETENTION PERIOD
HOT	ms	\$\$\$\$	\$	—
COOL	ms	\$\$\$	\$\$	30 DAYS
COLD	ms	\$\$	\$\$\$	90 DAYS
ARCHIVE	Hours	\$	\$\$\$\$	180 DAYS

ONLINE  
OFFLINE

or else there will be fine associated with it

- So this fine is called as early deletion fee



4. We can also change default access tier to cold...by going into configuration

The screenshot shows the 'Storage v2 (general purpose v2)' configuration page. On the left, there's a sidebar with navigation links: Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, Storage browser, Data storage (Containers, File shares, Queues, Tables), Security + networking (Networking, Access keys, Shared access signature, Encryption, Microsoft Defender for Cloud), and Data management.

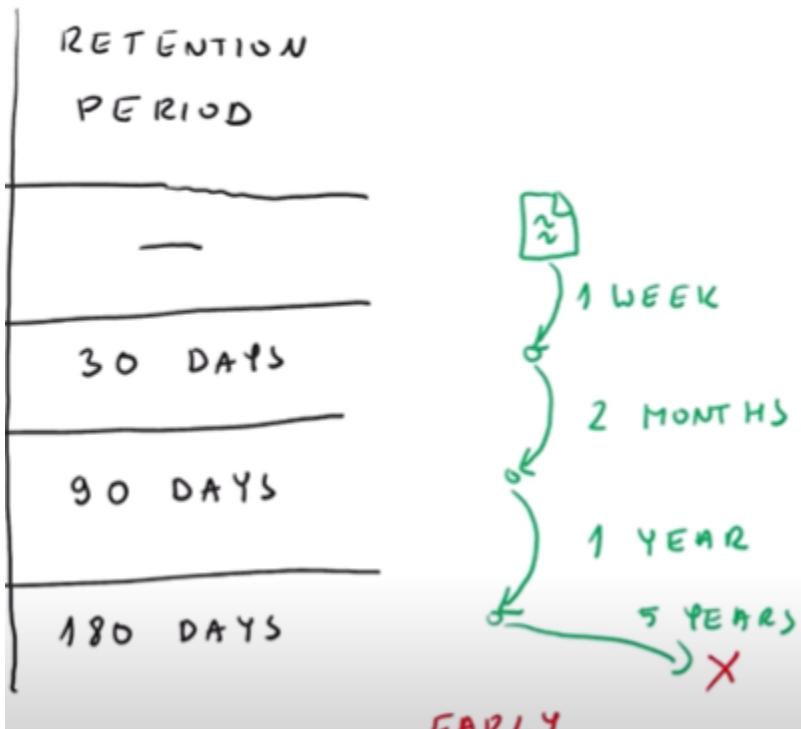
The main content area is titled 'Performance' and includes the following settings:

- Standard (radio button selected)
- Premium (radio button unselected)
- A note: "This setting cannot be changed after the storage account is created."

Other settings include:  
Secure transfer required (Enabled)  
Allow Blob anonymous access (Disabled)  
Allow storage account key access (Enabled)  
Allow recommended upper limit for shared access signature (SAS) expiry interval (Disabled)  
Default to Azure Active Directory authorization in the Azure portal (Disabled)  
Minimum TLS version (Version 1.2)  
Permitted scope for copy operations (preview) (From any storage account)  
Blob access tier (default) (Cool (selected))  
Large file shares (Disabled)

Life cycle management

- So we can define life cycle rules for files..to automatically change the access tiers



- Inside the datalake we have life cycle management

Home > dp203datalaketybuldemo\_1693725416796 | Overview

### dp203datalaketybuldemo

Storage account

Search  Upload Open in Explorer Delete Move Refresh Open in mobile CLI / PS Feedback

Essentials

- Resource group (move) : DP-203
- Location : West Europe
- Primary/Secondary Location : Primary: West Europe, Secondary: North Europe
- Subscription (move) : Visual Studio Enterprise Subscription
- Subscription ID : 1c461dd7-4fb5-4d22-b200-e052f75b9c6f
- Disk state : Primary: Available, Secondary: Available
- Tags (edit) : Add tags

Properties Monitoring Capabilities (5) Recommendations (0) Tutorials Tools + SDKs

Data Lake Storage

Hierarchical namespace	Enabled	Require secure transfer for REST API operations	Enabled
Default access tier	Cool	Storage account key access	Enabled
Blob anonymous access	Disabled	Minimum TLS version	Version 1.2
Blob soft delete	Enabled (7 days)		
Container soft delete	Enabled (7 days)		
Versioning	Disabled		
Change feed	Disabled		
NFS v3	Disabled		

Security

File service

31:22 / 40:54 • Lifecycle management

HD

### 3. Here we can add rules here

Home > dp203datalaketybuldemo\_1693725416796 | Overview > dp203datalaketybuldemo | Lifecycle management >

#### Add a rule ...

**1 Details**    **2 Base blobs**

A rule is made up of one or more conditions and actions that apply to the entire storage account. Optionally, specify that rules will apply to particular blobs by limiting with filters.

Rule name \*

Rule1

Rule scope \*

- Apply rule to all blobs in your storage account
- Limit blobs with filters

Blob type \*

- Block blobs
- Append blobs

Blob subtype \*

- Base blobs
- Snapshots
- Versions

**1 Details**    **2 Base blobs**

Lifecycle management uses your rules to automatically move blobs to cooler tiers or to delete them. If you create multiple rules, the associated actions must be implemented in tier order (from hot to cool storage, then archive, then deletion).

If

Base blobs were \*

Last modified  
 Created

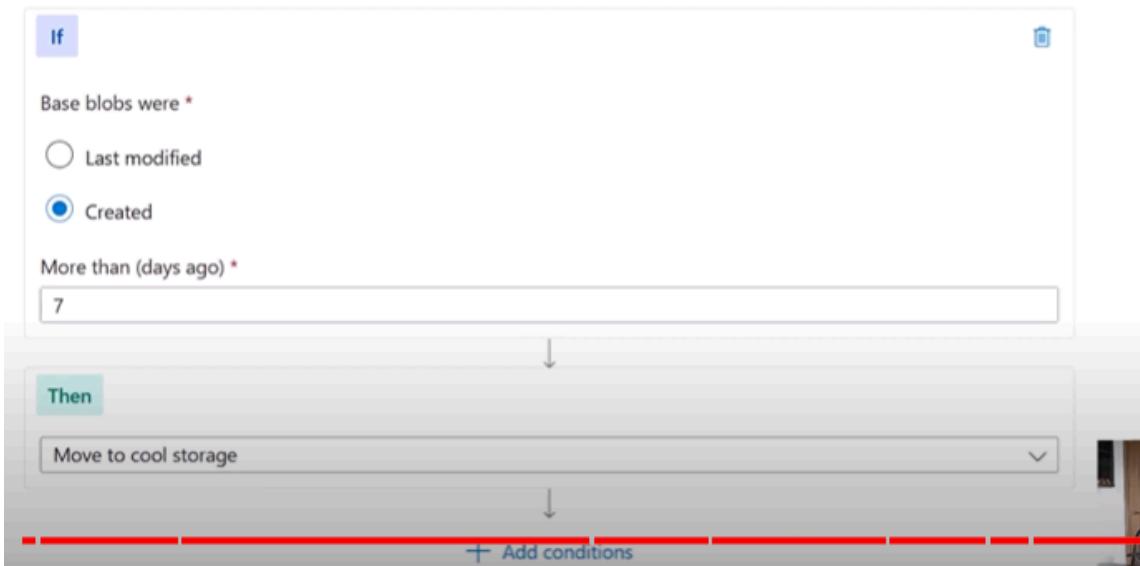
More than (days ago) \*

7

Then

Move to cool storage

+ Add conditions



So..here after the file is created...after 7 days it moves to access tier

4. And if we enable access tracking...then we can used it to change the access tiers

Details    2 Base blobs

Lifecycle management uses your rules to automatically move blobs to cooler tiers or to delete them. If you create multiple rules, the associated actions must be implemented in tier order (from hot to cool storage, then archive, then deletion).

If

Base blobs were \*

Last modified  
 Created  
 Last accessed

More than (days ago) \*

Enter a value

Then

Delete the blob



Azure Pricing Calculator

## 1. Here we can calculate the price using

The screenshot shows the Azure Storage Pricing calculator. At the top, there are four dropdown menus: Region (West Europe), Type (Data Lake Storage Gen2), Tier (Standard), and Storage Account Type (General Purpose V2). Below these are two more dropdowns: Access tier (Hot) and Redundancy (LRS). Under the heading 'Capacity', there is another set of dropdowns for Region, Type, Tier, and Storage Account Type, followed by a dropdown for File Structure (Hierarchical Namespace). A large input field shows '1000 GB'. Under 'Savings Options', it says 'Save up to 38% on pay-as-you-go prices with 1-year or 3-year Azure Storage Reserved Capacity.' There are two radio buttons: 'Pay as you go' (selected) and 'Pay as you go'. Below this is a section for 'Reserved instances' with radio buttons for '1 year reserved' and '3 year reserved'. A price of '\$19.60' is shown, with a note: 'Average per month (\$0.00 charged upfront)'. To the right of the calculator is a small photo of a man with glasses and a beard.

## 2. We will paying for capacity and transactions too

The screenshot shows the Azure Storage Pricing calculator with a different configuration. The top dropdowns are: Region (West Europe), Type (Data Lake Storage Gen2), Tier (Standard), and Storage Account Type (General Purpose V2). The access tier is now 'Cold' and the redundancy is 'LRS'. Under 'Capacity', the input field shows '1000 GB'. To the right of the input field are two buttons: one with a minus sign and one with a plus sign, both with a '\$4.50' label. Below the capacity section is a section for 'Transactions' with a plus sign button and a '\$3.64' label. At the bottom, there is a section for 'Other Operations and Meta Data Storage Meters' with a plus sign button and a '\$33.25' label. At the very bottom, there are two summary lines: 'Upfront cost \$0.00' and 'Monthly cost \$41.39'.

## 3. We can just play around with it

## DP-203: 06 - Common File Types (CSV, XML, JSON)

### CSV files

1. They are comma separated values
2. First row would be header..where it stores col names

C  
S  
V

COMMA  
SEPARATED  
VALUES

- TEXT FILES
- CAN BE OPENED BY ANY TOOL
- SIMPLE

3. And here there's no way we can enforce schema
4. Here every value will be a string inside the csv

	new 1
1	Id,Name,Title,Salary,Birthdate
2	1,Piotr,Architect,100.5,1980-02-01
3	2,Tom,Data engineer,80,1998-05-04
4	
5	

5. Sample CSV data
6. We have to ask metadata (to know the date format etc)

- ASK FOR METADATA:
- DATE FORMAT YYYY-MM-DD

C  
S  
V

COMMA  
SEPARATED  
VALUES

- TEXT FILES
- CAN BE OPENED BY ANY TOOL
- SIMPLE
- NO SCHEMA ENFORCEMENT
- EVERYTHING IS A STRING
- ASK FOR METADATA:
  - DATE FORMAT YYYY-MM-DD
  - TIME ZONE
  - NUMBER FORMAT

7.

8. Actually in CSV the data stored in flat

C  
S  
V

COMMA  
SEPARATED  
VALUES

- TEXT FILES
- CAN BE OPENED BY ANY TOOL
- SIMPLE
- NO SCHEMA ENFORCEMENT
- EVERYTHING IS A STRING
- ASK FOR METADATA:
  - DATE FORMAT YYYY-MM-DD
  - TIME ZONE
  - NUMBER FORMAT
  - MAX LENGTH
  - SCALE, PRECISION
  - COLUMN DELIMITER
  - TEXT QUALIFIER
  - ESCAPE CHARACTER
  - ENCODING
- FLAT

## XML

1. It stands for extensible markup language

```
<root>
...
*</root>
```

2. Here we work with tags
3. It is mostly used when to exchange data between diff application/systems

```
<root>
...
<order Id="1">
    <header>
        <OrderDate>2023-08-12</OrderDate>
        <Comments>Super shop!</Comments>
    </header>
    <lines>
        <line>
            <productId>1</productId>
            <quantity>3</quantity>
            <unitPrice>10</unitPrice>
        </line>
        <line>
            <productId>2</productId>
            <quantity>1</quantity>
            <unitPrice>100</unitPrice>
        </line>
        <line>
            <productId>34</productId>
            <quantity>39</quantity>
            <unitPrice>1</unitPrice>
        </line>
    </lines>
</order>
```

4. It supports hierarchical data
5. The XML code is well formatted and it is used by API's
6. But XML is very lengthy

## JSON

1. Stands for Java Script object notation

2. Its hierarchical and easy to understand

```
{  
  "Order": [  
    {"Id": 1,  
     "Header": {  
       "OrderDate": "2023-01-01",  
       "Comments": "Super product!"  
     },  
     "Lines": [  
       {  
         "productId": 1,  
         "quantity": 3,  
         "unitPrice": 10  
       },  
       {  
         "productId": 2,  
         "quantity": 31,  
         "unitPrice": 1  
       },  
       {  
         "productId": 3,  
         "quantity": 12,  
         "unitPrice": 10  
       }  
     ]  
   ]  
}
```

JAVA  
SCRIPT  
OBJECT  
NOTATION

- HIERARCHICAL
- EASY TO UNDERSTAND
- USED BY APIs
- NO TAGS
- LIGHTWEIGHT
- JSON SCHEM

3.

## DP-203: 07 - Parquet File Format

### Parquet Overview

1. It is most often used in data analytic/engineering
2. Parquet follows binary format and it requires some tools to open/work this files
3. So we have to download parquet viewer to open this files

07 - Common file types - PARQUET.png	01.10.2023 10:51	PNG File	334 KB
ParquetViewer_Signed.exe	29.09.2023 19:03	Application	72 396 KB
sample.parquet	30.09.2023 15:21	PARQUET File	3 KB

4. Here we can see the parquet file ...opened using tool

The screenshot shows the ParquetViewer application interface. At the top, there's a file list with three items: '07 - Common file types - PARQUET.png', 'ParquetViewer\_Signed.exe', and 'sample.parquet'. Below the file list is a window titled 'File: E:\Camtasia\Output\DP-203\07 - Common file types - parquet\sample.parquet'. The window contains a table with 10 rows of employee data. The columns are: id, firstName, middleName, lastName, gender, birthDate, ssn, and salary. The first row is highlighted with a blue selection bar. A red box highlights the entire table area. At the bottom of the window, status bars show 'Showing: 10 Results' and 'Loaded: 0 to 10 Out of: 10'.

	id	firstName	middleName	lastName	gender	birthDate	ssn	salary
▶	3766824	Hisako	Isabella	Malitrott	F	12.02.1961 05:00	938-80-1874	58862
	3766825	Dairy	Merissa	Fibben	F	19.05.1998 04:00	971-14-3755	66221
	3766826	Caren	Blossom	Henner	F	06.08.1962 04:00	954-19-8973	54376
	3766827	Darleen	Gertie	Goodinson	F	12.03.1980 05:00	981-65-5269	69954
	3766828	Kyle	Lu	Habben	F	15.02.1974 04:00	936-95-3240	56681
	3766829	Melia	Kristy	Bonhill	F	13.09.1970 04:00	960-91-9232	73995
	3766830	Yevette	Faye	Bebbell	F	07.09.1972 04:00	987-72-3701	92888
	3766831	Delpha	Kenisha	Gillison	F	25.06.1979 04:00	962-66-5404	51206
	3766832	Mikaela	Jenifer	Hallan	F	23.05.1973 04:00	911-38-3114	98887
	3766833	Cindi	Renita	Cousin	F	19.03.1979 05:00	666-50-3216	63646

### Storage modes

1. Parquet stores the data in hybrid format

	A	B	C	D
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5

2. Lets consider we have this table
  3. And assume A->Country, C->Sales Amount Data

## PARQUET:

- OPEN FORMAT
  - BINARY FORMAT → NEED A TOOL
  - HYBRID STORAGE
  - IDEAL FOR OLAP QUERIES

COUNTRY		SALES AMOUNT		..
*	A	B	C	D
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
..				

4.

```
SELECT  
    SUM(SALES_AMOUNT)  
FROM  
    TABLE
```

GROUP BY  
COUNTRY

5. Sample OLAP query
  6. Here in our query we only need 2 columns...A & C

### 3 Types of Storage

#### Row Storage

Here first we write A1,B1,C1,D1...and then A2,B2,C2,D2

1. And here we only need 2 col for our query..

#### 1. ROW STORAGE :

A1	B1	C1	D1	A2	B2	C2	D2
A3	B3	C3	D3	...			

2. But the main drawback is ...row storage..will read all the col's...which needs more compute power

#### Columnar Storage

1. Here the data will store in columnar format

#### 2. COLUMNAR STORAGE :

A1, A2, A3, A4, A5,	B1, B2, B3, B4, B5,
C1, C2, C3, C4, C5,	...

2. As our query only need col A and Col C...Now here we'll only be retrieving col A and Col C...and we ignore Col B ...in this storage format
3. It gives results much faster...this storage is great for OLAP

#### Hybrid Storage

SELECT

SUM(SALES\_AMOUNT)

FROM

TABLE

WHERE PRODUCT = XX

GROUP BY

COUNTRY

1. Lets modify our query

2. Here we have added a filter(where) product

	COLUMNS			
	COUNTRY	SALES AMOUNT	PRODUCT	
	A	B	C	D
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
.				

✓ ↘  
✓ ↘  
2

3. It combines the both row and columnar storage

4. Here in the hybrid storage..we'll be having row groups

	COLUMNS			
	COUNTRY	SALES AMOUNT	PRODUCT	
	A	B	C	D
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
.				

✓ ↘  
✓ ↘  
2

stored in columnar format

and inside row groups..the data is

### 3. HYBRID STORAGE

RG1 - A1, A2, B1, B2, C1, C2, D1, D2

RG2 - A3, A4, B3, B4, C3, C4, D3, D4

RG3 - A5, A6, B5, B6, ...

5. The data is stored in this way
6. Also Parquet stores the meta data ...like max,min values of col
7. And based on metadata it can filter row groups

### 3. HYBRID STORAGE

RG1 - A1, A2, B1, B2, C1, C2, D1, D2

~~RG2 - A3, A4, B3, B4, C3, C4, D3, D4~~

RG3 - A5, A6, B5, B6, ...

8. So this hybrid format is the most optimized one...it makes read operation very fast

### PARQUET:

- OPEN FORMAT
- BINARY FORMAT → NEED A TOOL
- HYBRID STORAGE
  - IDEAL FOR OLAP QUERIES
  - PROJECTION
  - PREDICATE

### Compression on Parquet

1. In our datalake the csv file takes 713MB and the same file in parquet takes 224MB

2. This because parquet follows dictionary encoding

<u>COUNTRY :</u>	<u>DICTIONARY</u>
USA 0	USA → 0
USA 0	POLAND → 1
POLAND 1	GERMANY → 2
POLAND 1	:
POLAND 1	:
GERMANY 2	
GERMANY 2	
⋮	

<u>COUNTRY :</u>
USA 0 0,2
USA 0
POLAND 1 1,3
POLAND 1
POLAND 1
GERMANY 2 2,2
GERMANY 2
⋮

3. We also have run-length encoding      :      :  
 4. Also the schema will be embedded in the parquet files

```
Cmd 2
Load parquet file to a data frame

1 df = spark.read.format("parquet").load('abfss://data@dp203datalaketybulddemo.dfs.core.windows.net/parquet')

▶ (1) Spark Jobs
▼ df: pyspark.sql.DataFrame
  - id: integer
  - firstName: string
  - middleName: string
  + lastName: string
  gender: string
  birthDate: timestamp
  ssn: string
  salary: integer
```

## 5. Same command for csv file format

Show that for CSV everything is a string

```
1 df2 = spark.read.format("csv").load('abfss://data@dp203datalaketybuldemo.dfs.core.windows.net/csv')

▶ (1) Spark Jobs
  ▾ df2: pyspark.sql.dataframe.DataFrame
    _c0: string
    _c1: string
    _c2: string
    _c3: string
    _c4: string
    _c5: string
```

## DP-203: 08 - Delta File Format

### Overview

1. This is the ultimate file format which has super cool features
2. Its an open source file format
3. Its same like parquet format...but it has transactional logs on top of it
4. Lets create a delta table and see how it is in data lake

### Create Delta table using SQL

```
1 CREATE TABLE IF NOT EXISTS people
2 USING DELTA
3 LOCATION 'abfss://data@dp203datalaketybuldemo.dfs.core.windows.net/Delta'
4 AS SELECT * FROM PARQUET.'abfss://data@dp203datalaketybuldemo.dfs.core.windows.net/parquet'
5

▶ (7) Spark Jobs
```

## 5. Here the important thing is our logs

The screenshot shows the Azure Storage Blob container interface. At the top, there's a search bar and a breadcrumb navigation path: DP-203 > dp203datalaketybuldemo | Containers. Below the path are standard file operations: Upload, Add Directory, Refresh, Rename, Delete, Change tier, Acquire lease, Break lease, and Give feedback. The authentication method is set to Access key (Switch to Azure AD User Account). The location is data / Delta. A search bar below the path contains the prefix 'blooms'. The main area displays a table of files:

Name	Modified	Access tier
...		
<b>_delta_log</b>	10/8/2023, 9:26:37 AM	Cool (Inferre
part-00000-e1f096c2-6d2b-408e-b2ab-49d551986178-c000.snappy.parquet		
part-00001-c476d810-78c0-4b2e-a1c2-4ba6a5638427-c000.snappy.parquet		

it stores the logs in the JSON format

The screenshot shows the Azure Storage Blob container interface, similar to the previous one but with a different path: DP-203 > dp203datalaketybuldemo | Containers > data / Delta / \_delta\_log. The table of files shows:

Name	Modified	Access tier
...		
<b>00000000000000000000.json</b>	10/8/2023, 9:26:46 AM	Cool (Inferre
_tmp_path_dir		
00000000000000000000.crc		

Below the table, there are tabs for Overview, Versions, Edit, and Generate SAS. The Edit tab is selected. A code editor window shows the JSON content of the selected file:

```

1  {"commitInfo":{"timestamp":1696749998451,"userId":"2099708230325839","userName":"piotr@tybulonazure.pl","operation":"CREA
2  {"metaData":{"id":"202952ab-2fac-4501-b84d-c0ddd378a68b","format":{"provider":"parquet","options":{},"schemaString":"\"
3  {"protocol":{"minReaderVersion":1,"minWriterVersion":2}}
4  {"add":{"path":"part-00000-e1f096c2-6d2b-408e-b2ab-49d551986178-c000.snappy.parquet","partitionValues":{}, "size":13407643
5  {"add":{"path":"part-00001-c476d810-78c0-4b2e-a1c2-4ba6a5638427-c000.snappy.parquet","partitionValues":{}, "size":10083206
6

```

## 6.

## 7. Here we have min,max values for every column

The screenshot shows the Azure Storage Blob container interface, similar to the previous ones. The table of files shows:

Name	Modified	Access tier
...		
<b>00000000000000000000.json</b>	10/8/2023, 9:26:46 AM	Cool (Inferre
_tmp_path_dir		
00000000000000000000.crc		

Below the table, there are tabs for Overview, Versions, Edit, and Generate SAS. The Edit tab is selected. A code editor window shows the JSON content of the selected file:

```

1  {"description":null,"isManaged":false,"properties":{},"statsOnLoad":false}, "notebook":{"notebookId":345664503074905
2  teger","nullable":true,"metadata":{}}, {"name": "firstName", "type": "string", "nullable": true, "metadata": {}},
3  :{ "numRecords": 5674560, "minValues": { "id": 1, "firstName": "Aaron", "middleName": "Aaron", "lastName": "A'Barro
4  :{ "numRecords": 4325440, "minValues": { "id": 5674561, "firstName": "Aaron", "middleName": "Aaron", "lastName": "A'
5  6

```

8. Now here we are changing the first name

```
1 UPDATE
2   people
3   SET
4     FirstName = 'Piotr'
5 WHERE
6   firstName = 'Pennie'
```

9. Delta table records this operation in data lake in Logs

```
1 UPDATE
2   people
3   SET
4     FirstName = 'Piotr'
5 WHERE
6   firstName = 'Pennie'

Cancel <-- Rewriting 2 files for UPDATE operation
▶ (3) Spark Jobs
```

10. Also it creates different files too

Name	Modified	Access tier	Archive status
_delta_log	10/8/2023, 9:31:38 AM	Cool (Inferred)	
part-00000-7da485ce-165d-4a0f-b352-b074249dc954-c000.snappy.parquet	10/8/2023, 9:26:37 AM	Cool (Inferred)	
part-00000-e1f096c2-6d2b-408e-b2ab-49d551986178-c000.snappy.parquet	10/8/2023, 9:26:31 AM	Cool (Inferred)	
part-00001-ef272658-1db5-4673-ba1e-91b2c129ca2d-c000.snappy.parquet	10/8/2023, 9:31:32 AM	Cool (Inferred)	

11. Now we'll again update the firstname

```
Cmd 6
1 UPDATE
2   people
3   SET
4     FirstName = 'Andrew'
5 WHERE
6   id = 1234

Cancel <-- Rewriting 1 files for UPDATE operation
▶ (4) Spark Jobs
```

12. For every CRUD operation...it stores the logs in delta logs and also it creates a parquet file as well

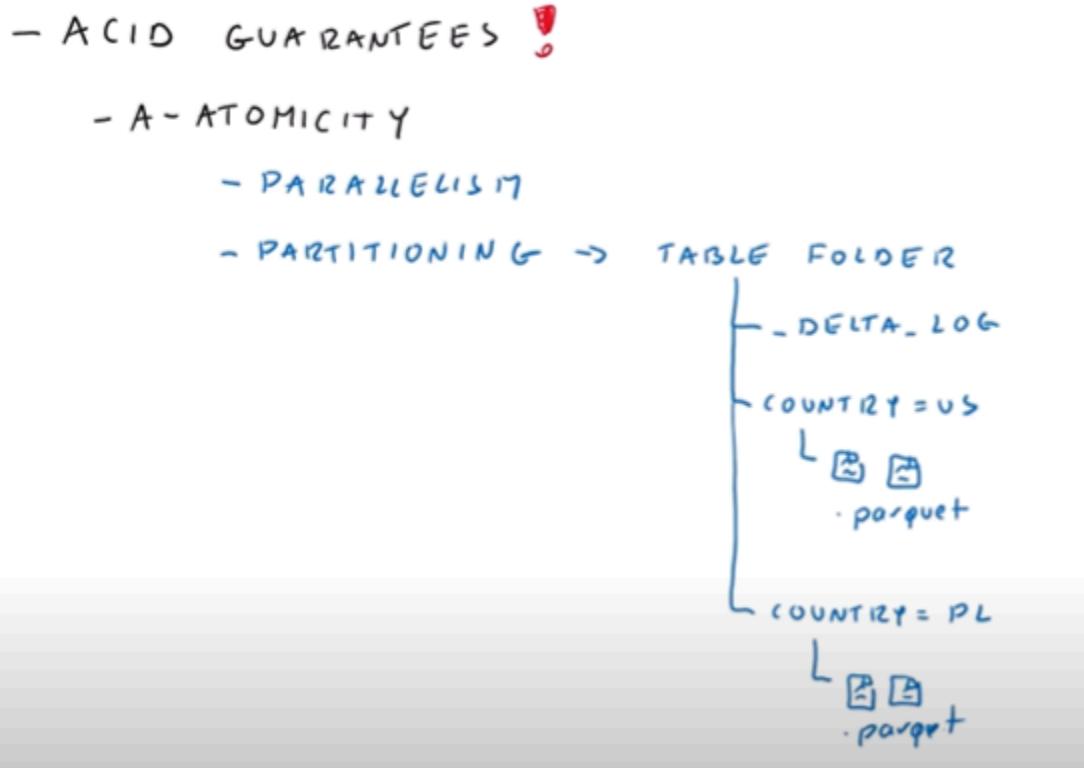
13. And here for every update it create a parquet file and in turn it is taking some storage...and at the end we have to pay for it

	Modified	Access tier	Archive status	Blob type	Size	Last modified
51c89-08fd-43e2-9447-129f23f49096-c000.snappy.parquet	10/8/2023, 9:34:10 AM	Cool (Inferred)		Block blob	127.87 MiB	Av
485ce-165d-4a0f-b352-b074249dc954-c000.snappy.parquet	10/8/2023, 9:31:38 AM	Cool (Inferred)		Block blob	127.87 MiB	Av
296c2-6db2-408e-b2ab-49d551986178-c000.snappy.parquet	10/8/2023, 9:26:37 AM	Cool (Inferred)		Block blob	127.87 MiB	Av
6d810-78c0-4b2e-a1c2-4ba6a553b427-c000.snappy.parquet	10/8/2023, 9:26:31 AM	Cool (Inferred)		Block blob	96.16 MiB	Av
72658-1db5-4673-ba1e-91b2c129ca2d-c000.snappy.parquet	10/8/2023, 9:31:32 AM	Cool (Inferred)		Block blob	96.16 MiB	Av

14. What's the point of storing historical data ..instead of updating in place

## ACID Guarantees

1. This delta lake guarantees the acid
2. Lets say we have many tables partitioned in different folders



3. And imagine we have created a query which updates every file
4. Now what atomicity guarantees..after making changes to every file then only it create delta log in our storage
5. C - Consistency

- C - CONSISTENCY -

```
graph LR; S1((STATE 1)) --> S2((STATE 2))
```

6. suppose are changing our data from state1 to state2...what consistency guarantee's is it will check the constraints and make sure they follow constraints of the table ....
7. Out of context example on consistency

Consider the following scenario without consistency checks:

- Account A has \$100, and account B has \$50.
- A user attempts to transfer \$150 from A to B (exceeding the balance in A).

If the transaction isn't checked for consistency, it might incorrectly update the balances as follows:

- Account A: -\$50 (negative balance)
- Account B: \$200 (total money increased)

- I - ISOLATION - OPTIMISTIC

- SNAPSHOT READS

8. I - Isolation
9. Example

#### Scenario without Isolation:

1. John reads the inventory, sees 1 size 10 shoe available, and initiates a purchase.
2. Before John's transaction completes (updating the inventory), Jane also reads the quantity (1) and starts her purchase.
3. Both John and Jane's transactions might succeed, showing a confirmed purchase for each. However, there's only one pair of shoes!

#### Scenario with Isolation:

1. When John starts his purchase, the database locks the size 10 shoe record, preventing Jane from reading the same data at that moment.
2. John's transaction completes, deducting the quantity from the inventory (now 0).
3. The lock is released, and Jane can now read the updated inventory (0). Her purchase will fail due to insufficient stock.

## 10. D - Durability

Durability, within the ACID properties of database transactions, guarantees that once a transaction is committed (successfully completed), the changes made to the database are persisted permanently. This persistence ensures that the updates survive even in case of system failures like crashes, power outages, or hardware malfunctions.

our storage redundancy will ensure this

## Audit History

### 11. Delta lake also provides us audit history

1 DESCRIBE HISTORY people								
(1) Spark Jobs								
version	timestamp	userId	userName	operation	operationParameters			
2	2023-10-08T07:34:10.000+0000	2099708230325839	piotr@tybulonazure.pl	UPDATE	{"predicate": "[\"(id#1742 = 1234)\"]"}			
1	2023-10-08T07:31:38.000+0000	2099708230325839	piotr@tybulonazure.pl	UPDATE	{"predicate": "[\"(firstName#1079 = Pennie)\"]"}			
2	2023-10-08T07:26:38.000+0000	2099708230325839	piotr@tybulonazure.pl	CREATE TABLE AS SELECT	{"partitionBy": "[]", "description": null, "isManaged": "false"}			
3	2023-10-08T07:26:38.000+0000	2099708230325839	piotr@tybulonazure.pl	CREATE TABLE AS SELECT	{"partitionBy": "[]", "description": null, "isManaged": "false"}			
3 rows		1.70 seconds runtime						
Command took 1.70 seconds -- by piotr@tybulonazure.pl at 10/8/2023, 9:52:54 AM on Piotr (admin)'s Cluster								

### 12. Now we can see who did what and all using the describe command

## Time Travel

1. Here we can see previous state of data...
2. Using the files delta lake created and using the delta logs

<input type="checkbox"/>	_delta_log		
<input type="checkbox"/>	part-00000-3fe51c89-08fd-43e2-9447-129f23f49096-c000.snappy.parquet	10/8/2023, 9:34:10 AM	Cool (Inferred)
<input type="checkbox"/>	part-00000-7da485ce-165d-4a0f-b352-b074249dc954-c000.snappy.parquet	10/8/2023, 9:31:38 AM	Cool (Inferred)
<input type="checkbox"/>	part-00000-e1f096c2-6d2b-408e-b2ab-49d551986178-c000.snappy.parquet	10/8/2023, 9:26:37 AM	Cool (Inferred)
<input type="checkbox"/>	part-00001-c476d810-78c0-4b2e-a1c2-4ba6a5638427-c000.snappy.parquet	10/8/2023, 9:26:31 AM	Cool (Inferred)
<input type="checkbox"/>	part-00001-ef272658-1db5-4673-ba1e-91b2c129ca2d-c000.snappy.parquet	10/8/2023, 9:31:32 AM	Cool (Inferred)

we can see the data of file at specific point of time

3. To time travel data we can use this command

### Time travel

```
1 SELECT * FROM people TIMESTAMP AS OF '2023-10-08T07:31:38.000+0000'
2 WHERE
3   | id = 1234
```

▶ (2) Spark Jobs

Table +

	<b>id</b>	<b>firstName</b>	<b>middleName</b>	<b>lastName</b>	<b>gender</b>	<b>birthDate</b>	<b>ssn</b>	<b>salary</b>
1	1234	Bambi	Britni	Nuccii	F	1981-03-02T05:00:00.000+0000	666 44 8021	52424

4. Also we can use this versions too..to see our previous state of data

```
1 DESCRIBE HISTORY people
```

▶ (1) Spark Jobs

Table +

	<b>version</b>	<b>timestamp</b>	<b>userId</b>	<b>userName</b>	<b>operation</b>	<b>operationParameters</b>
1	2	2023-10-08T07:34:10.000+0000	2099708230325839	piotr@tybulonazure.pl	UPDATE	["predicate": "[(id#1742 = 1234)]"]
2	1	2023-10-08T07:31:38.000+0000	2099708230325839	piotr@tybulonazure.pl	UPDATE	["predicate": "[(firstName#1079 = Pennie)]"]
3	0	2023-10-08T07:26:38.000+0000	2099708230325839	piotr@tybulonazure.pl	CREATE TABLE AS SELECT	["partitionBy": "[]", "description": null, "isManaged": "false"]

```
1 SELECT * FROM people VERSION AS OF 1
```

```
2 WHERE
```

```
3   | id = 1234
```

▶ (2) Spark Jobs

Table +

	<b>id</b>	<b>firstName</b>	<b>middleName</b>	<b>lastName</b>	<b>gender</b>	<b>birthDate</b>
1	1234	Bambi	Britni	Nuccii	F	1981-03-02T05:00:00.000+0000

5. This allows us to see and compare data  
 6. If we accidentally change the data..then we use this feature and go back to previous state

### Vacuuming operation

1. Here our delta lake created 5 files..

Location: data / Delta

Search blobs by prefix (case-sensitive)

Name	Modified	Access tier	Ar
<input type="checkbox"/> delta.log			
<input type="checkbox"/> part-00000-3fe51c89-08fd-43e2-9447-129f23f49096-c000.snappy.parquet	10/8/2023, 9:34:10 AM	Cool (Inferred)	
<input type="checkbox"/> part-00000-7da485ce-165d-4a0f-b352-b074249dc954-c000.snappy.parquet	10/8/2023, 9:31:38 AM	Cool (Inferred)	
<input type="checkbox"/> part-00000-e1f096c2-6d2b-408e-b2ab-49d551986178-c000.snappy.parquet	10/8/2023, 9:26:37 AM	Cool (Inferred)	
<input type="checkbox"/> part-00001-c476d810-78c0-4b2e-a1c2-4ba6a5638427-c000.snappy.parquet	10/8/2023, 9:26:31 AM	Cool (Inferred)	
<input type="checkbox"/> part-00001-ef272658-1db5-4673-ba1e-91b2c129ca2d-c000.snappy.parquet	10/8/2023, 9:31:32 AM	Cool (Inferred)	

And we'd like to remove extra files of data which we are no longer using

2. And also if our transaction is failed then it created additional files..and we'd like to remove them

Location: data / Delta

Search blobs by prefix (case-sensitive)

Name	Modified	Access tier	Ar
<input type="checkbox"/> delta.log			
<input type="checkbox"/> part-00000-3fe51c89-08fd-43e2-9447-129f23f49096-c000.snappy.parquet	10/8/2023, 9:34:10 AM	Cool (Inferred)	
<input type="checkbox"/> part-00000-7da485ce-165d-4a0f-b352-b074249dc954-c000.snappy.parquet	10/8/2023, 9:31:38 AM	Cool (Inferred)	
<input type="checkbox"/> part-00000-e1f096c2-6d2b-408e-b2ab-49d551986178-c000.snappy.parquet	10/8/2023, 9:26:37 AM	Cool (Inferred)	
<input type="checkbox"/> part-00001-c476d810-78c0-4b2e-a1c2-4ba6a5638427-c000.snappy.parquet	10/8/2023, 9:26:31 AM	Cool (Inferred)	
<input type="checkbox"/> part-00001-ef272658-1db5-4673-ba1e-91b2c129ca2d-c000.snappy.parquet	10/8/2023, 9:31:32 AM	Cool (Inferred)	

3. Vacuum is just for this....it will remove

- VACUUM

- UNCOMMITTED

- NO LONGER NEEDED (RETENTION)

by default delta

lake store historical files for 7 days...then later if we perform vacuum ..then old files will be removed

4. If delete files using vacuum..then we cannot use time travel to go to prev state

## Schema enforcement

1. Sometimes the schema is not enforce in parquet format

first_name	age
leonard	51
bob	47
li	23

2. Here we have created a df with first name and age
3. Later we have appended new data with new column datatypes to the prev df

```

1  %python
2  columns = ["first_name", "favorite_color"]
3  data = [("sal", "red"), ("cat", "pink")]
4  rdd = spark.sparkContext.parallelize(data)
5  df = rdd.toDF(columns)
6
7  df.write.mode("append").format("parquet").save("tmp/parquet_table1")
8
9  spark.read.format("parquet").load("dbfs:/tmp/parquet_table1").show()

```

▶ (6) Spark Jobs

- ▼ df: pyspark.sql.dataframe.DataFrame

```

first_name: string
favorite_color: string

+-----+-----+
|first_name|favorite_color|
+-----+-----+
|  leonard|        NULL|
|    cat|         pink|
|   sal|          red|
|   bob|        NULL|
|    li|        NULL|
+-----+-----+

```

4. Then for the prev df ..we get null values (as prev it was age...now it is fav\_color)
5. Here we have corrupted our data
6. Coming to delta

7. Here we have create sample df using delta format

```
1 %python
2
3 columns = ["first_name", "age"]
4 data = [("bob", 47), ("li", 23), ("leonard", 51)]
5 rdd = spark.sparkContext.parallelize(data)
6 df = rdd.toDF(columns)
7
8 df.write.format("delta").save("dbfs:/tmp/delta_table1")
9
10 spark.read.format("delta").load("dbfs:/tmp/delta_table1").show()
```

▶ (11) Spark Jobs

▶ df: pyspark.sql.dataframe.DataFrame = [first\_name: string, age: long]

```
+-----+---+
|first_name|age|
+-----+---+
|  leonard| 51|
|    bob| 47|
|     li| 23|
+-----+---+
```

8. And we try to append 2nd df with diff schema

```
1 %python
2 columns = ["first_name", "favorite_color"]
3 data = [("sal", "red"), ("cat", "pink")]
4 rdd = spark.sparkContext.parallelize(data)
5 df = rdd.toDF(columns)
6
7 df.write.mode("append").format("delta").save("dbfs:/tmp/delta_table1")
8
9 spark.read.format("delta").load("dbfs:/tmp/delta_table1").show()
```

▶ (2) Spark Jobs

⊖**AnalysisException**: A schema mismatch detected when writing to the Delta table (Table ID: 82678b70-84ca-4e2c-  
To enable schema migration using DataFrameWriter or DataStreamWriter, please set:  
.option("mergeSchema", "true").  
For other operations, set the session configuration  
spark.databricks.delta.schema.autoMerge.enabled to "true". See the documentation  
specific to the operation for details.

Table schema:

```
root
-- first_name: string (nullable = true)
-- age: long (nullable = true)
```



then in delta we get analysis exception error

- SCHEMA ENFORCEMENT
  - PARQUET X
  - DELTA ✓

9.

## Checks and constraints

1. We can also define constraints like

### Checks and constraints

```
1 ALTER TABLE people ADD CONSTRAINT GenderCheck CHECK (gender IN ('F', 'M'))  
▶ (6) Spark Jobs  
OK  
Command took 3.69 seconds -- by piotr@tybulonazure.pl at 10/8/2023, 10:09:03 AM on Piotr (admin)'s Cluster
```

## Schema evolution

## 1. Here we can also merge schema

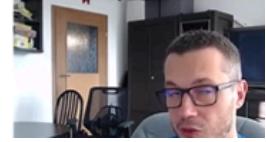
### Schema evolution with Delta

```
1  %python
2  columns = ["first_name", "favorite_color"]
3  data = [("sal", "red"), ("cat", "pink")]
4  rdd = spark.sparkContext.parallelize(data)
5  df = rdd.toDF(columns)
6
7  df.write.option("mergeSchema", "true").mode("append").format("delta").save("dbfs:/tmp/delta_table1")
8
9  spark.read.format("delta").load("dbfs:/tmp/delta_table1").show()
```

▶ (10) Spark Jobs

▶ df: pyspark.sql.dataframe.DataFrame = [first\_name: string, favorite\_color: string]

first_name	age	favorite_color
leonard	51	NULL
cat	NULL	pink
sal	NULL	red
bob	47	NULL
li	23	NULL



## 2. Previously we get an error..while we merging different schema's

### Merge Operation

#### 1. Here if there's new data..then we can merge it with old data

- MERGE

NEW DATA			
ID			
1			
2			

↓  
INSERT

MERGG

EXISTING DATA			
ID			
1	→ UPDATE		
3			
5			
7			

#### 2. Here if the new data id not present ..then in old data..then it insert the data

### Optimize and Z-order

1. Lets suppose we have multiple small delta files in our storage..
2. Then reading this all files or to perform any operation on all files...the time complexity increases to perform for all files
3. So optimize combines these small files into one big file

#### OPTIMIZE:

- **Purpose:** OPTIMIZE reorganizes data files within a Delta table to improve query performance. It focuses on two main aspects:
  - **File Sizing:** OPTIMIZE aims to create data files with balanced sizes. This can improve efficiency when reading data, as scans don't have to process large variations in file sizes.
  - **Vacuuming:** It can optionally remove unnecessary small files created during data updates. This helps to reduce overall storage footprint and potentially improve some queries.

#### Example:

Imagine a Delta table storing website clickstream data. Initially, the data might be scattered across many small files due to frequent updates. Running OPTIMIZE on this table would:

1. Analyze the existing data files and their sizes.
2. Coalesce smaller files into larger ones, aiming for a more balanced file size distribution.
3. Optionally, remove very small files that don't contribute significantly to data storage.

4. Z-ordering

#### Z-Ordering:

- **Purpose:** Z-Ordering rearranges data within files based on specific columns. It sorts rows within each data file based on the chosen column(s). This aims to colocate rows with similar values in the Z-Order columns within the same file segments.

#### Example:

Consider a Delta table storing user purchase data with columns like `user_id`, `product_id`, and `timestamp`. You frequently query for user purchases based on `user_id`. Z-Ordering this table by `user_id` would sort all purchase records for each user together within the data files.

#### Benefits:

When a query filters data based on the Z-Order column(s), Delta Lake can leverage data skipping. It can efficiently identify and skip irrelevant data files that don't contain rows matching the query predicate. This significantly reduces the amount of data scanned and improves query performance.

