Usage of data lakes and/or data warehouses

Céline Thorns

Seminar with Practical: Scalable Computing Systems and Applications in AI, Big Data and HPC

March 31, 2023

Contents

1 Introcution

Finally computers are able to perceive the world around them in nearly every component possible. However, handling this vast knowledge still comes with some errors. With the beginning of the new century the amount of data had increased. Additionally it also changed the form it could take. Because of this alteration data warehouse structures could no longer serve the demand and data lakes were introduced. In these data can take form in more than structured ways. They can be stored in semi-structures like json or xml, which still have a connection, but are not relational table blocks. Other data is even stored in an unstructured way. This loose structure allows faster uploads, without ordering or denormalizing these datafiles. However, this unordered data will take more resources when it is not only read but overwritten. This is why the purpose of data lakes in contrast to data warehouses is more focused on offering large data storage, not on offering permanent data quality.
This report explores the change of forms that databases take. It discusses the reasons behind data warehouses and the cause why data lakes have started to replace them. Nonetheless, with the continuous raise of permanent data exchange even data lakes suffer from their weaknesses more prevalent. The core of this report is about the next generation of databases: data lakehouses and one of their application delta lake. As such a section covers the mechanics, which enables reliable data while still archiving adequate performance. Examples of certain usecases are presented in code. In the end a short outlook examines other methods to raise performance on top of the format of databases.

2 Data Warehouses and Data Lakes

In the 1980s computers have archived enough performance and data storage to process and contain lager datasets. Data warehouses entire existence evolves around allowing data analysis from a centralized platform and with their strict rules allow wealthy management and performance opportunities. Their basic structure consists out of schemes for the data and transactions to preserve their reliability. This model requires a uniform build-up, which is why the data is first transformed, before it is possible to load it into the data base. Smaller outposts of this data warehouse are called data marts, which allows cheaper horizontal scaling of memory cells. OLAP (online analytics processing) servers are connected to data warehouses and function as access points for data researchers, which is the target demography of this structure. The separation between data warehouses and OLAP servers allows a divide between memory and processing units and by this safer environments for the stored data and faster processing. The strict structure enables minimal search and by this fast queries. However loading data into and maintaining the data center is expensive.[8]
In 2010 data lakes have been invented as a replacement for data warehouses. The reason for their invention is based on three problems from data warehouses. While data warehouses are outstanding for relational data, it cannot process non-structured data without forcing them into a fixed table. Second, the formating of data, as well as their cleaning and denormalization results in high cost for data sets. Lastly, data warehouses do not offer any support for data science tools or machine learning. [7]
Contrary to data warehouses data lakes are great for raw data and adapts them with APIs.

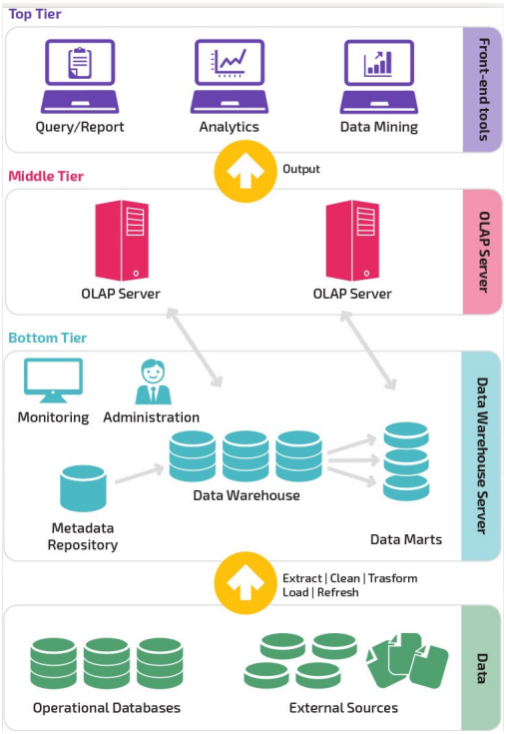

Figure 1: Structure of a data center

Instances for data lakes are S3 or HDFS. The open file formats allows connections with machine learning. Certain parts of the data lake, which fit into relational schemes, are extracted and loaded into data warehouses via ETL. [7]
As a result data lakes focus more on faster storage of large amounts of data than extensive queries of the stored data. Handeling data in all forms is possible, which is why when storing data, this data receives either minimal or no transformation at all. This allows entries outside of text-data access into the database. As an example a data lake can store images, audio or video data and references to other entries. With the rise of multiple different file-formats and larger storage capabilities of non-professional private computers, the amount of data outside of the strict data warehouse schema has increased in the last decades. By now 80% of all data is unstructured. This data can be stored from structures resembling a data warehouse, over semi-structured data following a non-relational construction, to unstructured and raw dumps. While unstructured data refers to data-files like audio or integrated structures, non-relational structures are for example graph, key-value, json and xml structures.
This difference in data leads to different intentions behind choosing the correct implementation. While data warehouses have a schema-on-write processing, which allows cheap and easy queries, fast information tracking and speedy changes in entries, the structure also requires knowledge of the setup, before integrating the data. Data lakes focuse on schema-on-read. In this schema data is not stored in columns, rather software archiving and reviving the data creates imaginary structures. [7]
Consequently this structure also allows high flexibility when adapting structures towards new projects. Schema evolution requires merely a change of the metadata. However, changes of structure and entries meanwhile are not documented and it is not possible to undo them.[7]

The disadvantage of loose structures is the omitting of adapted software. Additive content like SQL is not compatible with schema-on-read and fast search queries often only archive prognoses. [2]
Also, this ingenuous input of data might impact security. First, the lax structure when uploading data might incite someone to just dump everything into the database without insuring quality or reliability. This thread increases further downstream in the data lake, since these structures do not necessary follow the ACID principles.

2.1 ACID

ACID is an acronym for Atomicity, Consistency, Isolation and Durability and depicts important principles of transactions. Atomicity ensures a correct flow of transactions without hindrances or disturbances. For this each action is executed in one closed transaction. If an error occurs a rollback will be called and every variable will return to its previous value. So in the end either every desired change happens in one transaction or nothing changes at all.[8]
Consistency prevent changes, when they are incongruous with current constraint or required patterns. So if an entry does not have a key or fit into a schema. It also demands that no matter where and by whom a variable is read, it will have the same value.[8]
Isolation ensures concurrency over multiple variables and actions. It allows multiple users access to certain areas, while preventing multiple writing accesses at the same time. For this it can engage optimistic and pessimistic concurrency. Optimistic concurrency assume intersections between different orders are rare and by this allow every order to write. The system checks later if the consistency has been ensured and if not, restores previous settings. Pessimistic concurrency assume intersecting transactions are common and prevent any third party from intervening by placing locks on required entries till a transaction is finished. [7]
Durability ensures memory safety even in crashes. One way to do this is to generate a backup or to process overlapping information between multiple servers. However this might clash with consistency. Changes entail notifications to all servers with copies of information. These prolong any kind of action and messages from both sides can disappear or intersect with different messages from different times. Since time can differ between servers the correct order of messages might get lost or orders in total can disappear. [7]
Especially in data lakes the large parallelisation and unfamilitarity of the data disables greater power of each attribute. Most information about data is saved in metadata, which are expensive to search across.

3 Data Lakehouses and Delta Lake

In a survey by Fivetran [11]the biggest problem for data scientists is the lack of data quality and reliability. When asked, 60% of the respondent state, that the lack of data quality would be a challenge for them. Furthermore 86% of interviewees state to, at some point have worked with stale data, which was more than a couple of weeks overdue and 41% with data older than two months. Moreover 90% of the respondents admit, that they regularly work with unreliable data sources.

The Kaggle data scientist survey[10] focuses on the process flow of workers. In it data scientists state, that they spend most of their time analyzing and understanding data. However, some of the most frequent activities are understanding the service and infrastructure of a database.
Data warehouses enable business professionals to collect knowledge from other operational database systems. By using ETL (Extract, Transform, Load) he data is changed to fit into the schema of the database structure. This strict structure allows powerful maintaining and operating tools like SQL.[3]
However, data warehouses are unable to interpret other file formats like time series logs, images and unstructured documents.. Furthermore, the warehouse is constipated for human interaction and does not possess any API for machine learning and other data science tools, which require direct access. [3]
The invention of and change towards data lakes is often associated with the rise of big data. However, as seen above data warehouses already contained weaknesses, which have only increased with modern demands. Now, the situation continues to change in a permanent flux of rising demands for larger data, faster queries and additional content, so even data lakes have to adapt. One possible way is to combine advantages of both formats into a new one: a data lakehouse.[3]
Delta lake offers a suitable solution to the problems from above. The format surrounding

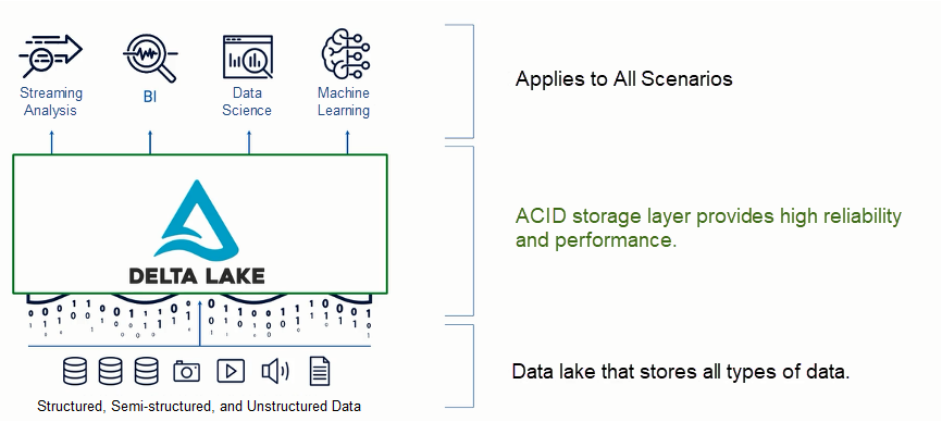

Figure 2: Structure of Delta Lake

delta lake is a data lakehouse with special interest towards ACID properties. It consists out of a two tier architecture, which is more complex than data lakes but allows more freedom than data warehouses. In one layer all the data is saved similar to data lakes with or without any binding schema and a wide range of formats. The data retain their open file format, allowing other users to directly access them.
The second layer stores metadata about the data of the first layer in an object store and enables the missing features of data lakes while enabling a better performance. This is possible because the metadata manages the modification of data.
Data Lakes have metadata too, which organizes structure and elements, however, they are not as strong as in lakehouses. Previously in data lakes, when a file is deleted, the metadata-files like parquet in connection to the file needs to be rewritten, before the file itself could

be deleted. Afterwards, any other file containing metadata about the deleted file has also to change its content. Any other copy of the file needs to be deleted. This can cause a problem, when another entity is reading the file, while it is deleted, as it causes a consistency issue. Additional, deleting a file can fail because of message failure or missing authorities and the complete database becomes incoherent.
However, Delta Lake's metadata layer contains one additional directory, the _data_log-file, which saves the changes as a write ahead log, including which files are part of each table. These directories are adapted with each transaction and describe the history of the data area. Duo to the additional format the data quality is enforced. Furthermore, since the logs contain information about each action, these logs allow time travel. When they are copied, it is possible to migrate a clone of the table without copying the data based on the log. Not only do they allow simpler change of data, they also enable larger transactions like merging and separations of the whole table. These features allow datalake programs like Delta Lake to be used in other areas like data busses. Additionally stale data is sorted in the background, which allows fresher data to work faster.
The reason behind the invention of Delta Lake was the growing discontent with many Spark readings and writings failing because of too many concurrent transactions intermingling with each other. 2019 the founders decided to release Delta Lake as open source software in order to ensure transactional protection for data used in Delta Lake. Delta Lake enables programming with SQL, Java, Python and Scala and allows intermingling between them with certain commands.
Other systems Apache Iceberg implement data lakehouses too. Delta Lake's advantages lie in its unlimited scaling capabilities as well as its diversity in use cases and file formats.

3.1 Transaction log

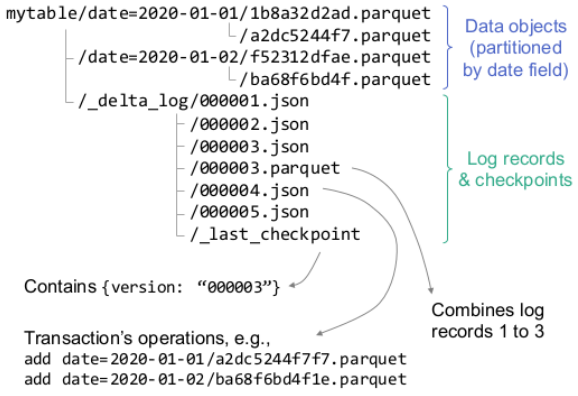

Figure 3: File directory of a delta lake table

Delta Lake possesses a file protocol build upon a transaction log. This formation allows higher features like transactionality and schema enforcement without the risk of inconsistency which data lakes risk.

When creating a table in Delta Lake, the system creates a setup like in figure x. An automated file management as a transaction log is constructed on top of the file containing the data. In it json-files describe table versions, in a way the directory knows how to read. With each commit of a snapshot from a user another json-file is created, which contains properties about the commit itself as well as the actions executed in this snapshot as metadata. Furthermore the log-files include table name, schema and partitioning. The json-file is build-up out of different actions like update metadata, add file, remove file, set transaction, change protocol and commit info.
While the first elements are self explanatory, removing files is not as easy as adding them. Files are not deleted imminent after a commit, which orders their deletion, but instead deleted only days afterwards. Set transaction allows structured streaming by containing the id of the stream, which also exists in the checkpoints of the stream, and the batch id, meaning the amount of committed batches can be determined. Delta Lake is upwards and downwards compatible. For this it utilizes change protocol action to read the metadata in the way of the current version. The commit information is saved in the last action. These features allow a faster and easier access to data and enforces data reliability.[4]
The transaction log acts as the single source of truth. It allows a safeguard of the current table, by reading the transaction log and executing the adding and removing the current table is created. When reading adds and deletions cancel each other out and reconstructing the table can proceed faster.[4]
Delta Lakes allows ACID features and as such each transaction requires isolation. Commitioners create a snapshot of the current table, before transmitting its changes to the table all at once. The creation of the corresponding json-files occurs atomically with the commit of the new data. [4]
Mutual exclusion requires and guarantees data reliability. In this methods even with multiple writers serialability is guaranteed, by every writer agreeing to the order of changes. The priority is given to the one who has read the latest json-files.[4]
Delta Lake employs optimistic concurrency. Since it uses the advantage of cheap data storage from data lakes, a vast amount of data entries are expected and as such updating the same file should be rare. In order to enforce this concurrency, each snapshot saves the start version and records all read and writes the user demands. The commit is denied if the start version is no longer concurrent. In this case the snapshot is checked for all the changes it demands and weather the variables are still up-to-date. If two writers try to delete the same variable the system throws an exception.[2]
Saving additional files for the transaction log costs vast amounts of storage space. In order to limit the extra space, Delta Lake uses Spark for scaling. To increase speed for readers at a certain commit interval Delta Lake creates a parquet-checkpoint, which contains the state of the table at present. Otherwise the danger of a bottleneck is imminent. Each object store will have thousands of data-files and the demand for the transaction logs is for every command necessary. With Spark the scaling is cut short, since each cluster is an individual table, which has the metadata saved separately.[2]
When computing the state of the table, Spark will cache the table based on the commits. If the data of the table changes, Spark integrates this new data into the snapshot with the help of listFrom. This function is a feature of the lock store and lists all changes so Spark can update the data just like in the beginning. If listFrom outputs a new checkpoint, the

cache-version will be discarded and the process begins anew. [5]
Furthermore the transaction log allows time travel. Time travel is possible based on versions and timestamps. For time travel based on versions the command is executed via _select from version_or with an @

```
SELECT * FROM table VERSION AS OF 0001;
SELECT * FROM table@0001;
spark.read.format("delta").option("versionAsOf",0001).load("/location") #for
    python
```

It is not possible to directly call from a timestamp. Instead the program has to find the version correspondent to the demanded timestamp in order to be able to call it. To achieve this the timestamp is used as a relation. [5]
Additionally using timestamps as indicator can create problems when the data is spread over multiple systems and they no longer act synchronous because of clock skew. As such it is not necessary true that a version with a later timestamp on one system is younger than the version with a younger timestamp.[5]
Another requirement for time travel is, that the log file and the files corresponding to it still exists. Because of memory limitations, these files only exists 7 to 30 days respectively with checkpoints getting deleted more aggressively. It is possible to change this time frame with the action

```
delta.logRetentionDuration=interval <interval> .
```

In the end time travel is intended to be just a solution to prevent mistakes. Computing all the undone transactions would not scale.[2]
Nowadays data is constantly evolving and changing. Delta Lake enforces schemes but also allows easy adaptation to completely disregard them. Schemes state the structure of the data while schema enforcement prevents non-compatible data from entering the table. In Spark every data frame possesses a schema, which is transmitted in Delta Lake and becomes a table in a json-file. Necessary for a schema in this situation is a list of fields, each possessing a name and type for entries as well as a boolean determining if entries can take the form of null. Schemes can be broken when a write operation tries to add entries with an additional column or different data type, but also since parquet is case sensitive, spelling errors. In this case the atomic transaction is rolled back and an exception is thrown. This schema enforcement is necessary for machine learning, data analysis and visualization tools. In order to adapt larger external structures, pipelines can build a multi-hop architecture in front of the entry. In it the different sections filter bad data, adapt the external schema to the inner and augment the data.[5]
Schema evolution is often used to change the data when data is either appended or overwritten. Again multiple methods exists, which enable schema evolution. Using

```
    spark.read.option("mergeSchema", "true").parquet("/location")
```

allows a soft changes, which only changes the schema but still maintain the already existing data. Example for this are adding columns or switching to bigger data-types like from integers to longs. Compared to this

```
spark.read.option("overwriteSchema", "true").parquet("/location")
```

the schema changes in a hard way and data loss is possible. The command overwrites the table and can be used to as an example drop a column. Lastly the order

```
spark.delta.schema.autoMerge=True.parquet("/location")
```

completely disregards any schema and no longer imposes schema enforcement. [5]
Delta Lake processes multiple connectors to other software like HIVE, Snowflake and Amazon Athena, which allows data from Delta Lake to migrate to other processors. It is possible to change data simply with the command

```
df1.toPandas()
```

into an Pandas dataframe. On the other side Delta Lake also has multiple partners like Google Dataproc, that easily enables Delta Lake to import Spark packages from parquet-files into Delta Lake.

3.2 Data manipulation language operations

Data manipulation language (DML) form a basis on which a program can insert delete and merge data. The following section discusses how Delta Lakes implement these orders and how it increases performance.
Adding new data requires Delta Lake to scan the files two times before possessing the ability to determine where the changes needs execution. The first scan is a complete scan, which is required to identify in which tables the data needs to change. This scan can be shortened by using bloom filters or z-order to exclude openly unfit tables. Parquet files aditionaly contain statistics like min/max-values of its entries, which allows faster searching. Larger changes to increase performance requires changes to the layout of the data table like multi-column sorting. [2]

3.3 Transaction log

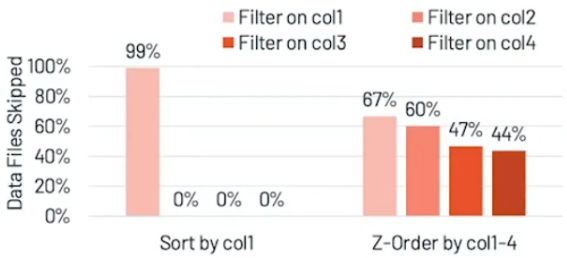

Figure 4: Reduce of searched files when using when using z-order[3]

Using the list of tables from the first scan, the second scan chooses the overdue data and rewrites them as a new file. The old file is marked with a tombstone but not deleted. This enables the time travel to reconstruct old data, while the tombstone prevents a break in concurrency between systems. Deleting works in a similar way, but in this case the system creates a new file without the selected data. With the order

```
table.vacuum()
```

the files are immediately permanently deleted.[6]
Merge is the most powerful tool in DML. It is based on the standard SQL syntax

```
table.merge(table2, "table.id = table2.id")\
    .whenMatchedUpdateAll()\
    .whenNotMatchedInsertAll()\
    .execute()
```

Again two scans are needed. The first determines the data parts for the inner join, the second for the outer join. Afterwards the data is updated, deleted or inserted into new files. In order to increase speed for the inner join the search space can be narrowed down like using shuffle partition or adjusting broadcast thresholds. Furthermore small files can be compacted into larger files. However in order to increase speed in the outer join, smaller data files, which fits into the cache doubles the speed for finding all elements.[6]

4 Coding with Delta Lake

After the initial set-up of Spark is required in which Delta Lake is started and a location for the storage is defined. Delta Lake enables reading of multiple formats like json-files or even images. Furthermore it allows a wide area of functions like transferring frameworks or extracting limits. It is possible to read and write files from a delta table. As a side note, the mode "write is only reserved to creating new files while the mode "overwrite" only allows changing to already existing files.
The biggest problems with data lake is its inability to handle queries, which turn up empty. While it makes sense to throw an exception if a user tries to change a variable, which does not exists, the empty search should prevent this case. As an example in the setup of the notebook if the images with height over 2000 pixels are deleted and later the order

```
fimage1.update(
  condition="image.height < 2000",
  set={"image.nChannels": "3" }
)
```

demands a change under a condition, the system will throw an exception.

5 Additional content

Armbrust et all [2] discusses in their paper different performances of Delta lake. In it Delta Lakes performs better than any other entity. Additional help like vectorized execution

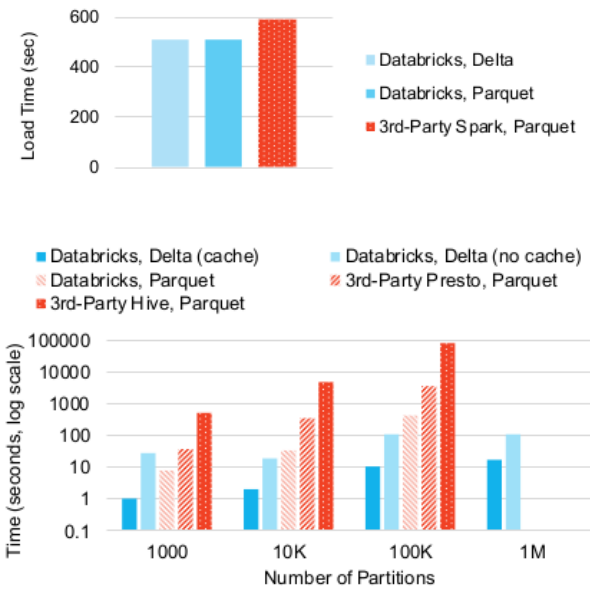

Figure 5: Left:Performance text to load 400 GB of TPC-DS storesales into Delta or Parquet files. Right: Time needed for different amounts of small partitions. Non-delta systems were unable to prcess over one million partitions in under an hour.[2]

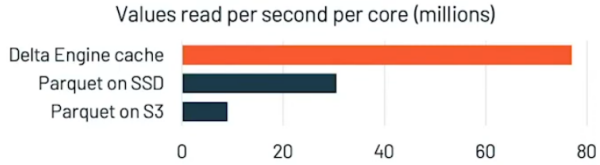

Figure 6: Difference in reading time for parquet files[3]

engines adapted for specific CPUs can increase its performance. This allows Delta Lake to read more values per second per core than SSD or S3 as shown in figure 6. Additionally other differences across formats may offer better performance or storage possibilities. Duo to horizontal scaling, the appointment of different data blogs and their composition changes the processing time of different queries. Since a suitable composition is depending on what queries are used often, a correct layout is often difficult to find. With the rise of AI or animal-inspired programs this overreaching problem is solved outside of the formats. [3]
The step from data warehouses to data lakes up to lakehouses are evolutionary differences, which changes the fundamental principle behind each format. Changing from one format into another requires extensive changes, so even when lakehouses offer faster and safer alternatives

for an existing structure, changing the formats might require more effort than gain in the foreseeable future. Furthermore just like with data lake it is to be expected, that one big data set will be split up into multiple formats. Other attempts like Apache Avro break away from classical schema too without adapting to any format.[2]

6 Outlook

The general idea behind Delta Lake is fascinating. By now, multiple aspects of this world can be implemented as a digital twin or collected into a group, which needs permanent updating. Since multiple instances are connected to one service provider, it intermingles with big data. Especially in areas of computer-human interaction, computers need to react immediately in a changing situation based on previous knowledge. Big data is no longer stale but consits out of a permanent stream of new data. It is not possible to dismiss Delta Lakes perfromance, when it is deployed in a fitting setting. However, Delta Lake has two problems: It is young and not beginner-friendly. While the multitude of languages allow an easy understanding for a large group of people and its multitude of predefined functions allow a smooth execution of certain tasks, bug-fixing becomes soon necessary and proves difficult. Already in the documentation errors occur. In case of overzealous error messages it is not possible to rely on a more experienced group, because the open source version has only opened to the public since 2019. Furthermore it should be possible to navigate in an lakehouse without detailed knowledge of the data, but Delta Lake demands extended knowledge or it will penalise you. Without any doubt Delta Lake will grow and with it these problems will pass on. Currently the problems can be softened when a group of people works on Delta Lake and the diversity enables better problem solving.

References

[1] ARMBRUST, Michael, et al. Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. In: Proceedings of CIDR. 2021. S. 8.

[2] ARMBRUST, Michael, et al. Delta lake: high-performance ACID table storage over cloud object stores. Proceedings of the VLDB Endowment, 2020, 13. Jg., Nr. 12, S. 3411-3424.

[3] Matei Zaharia; Lakehouse: A New Generation of Open Platforms for Data Warehousing and AI; Wrangle Summit; 2021

[4] Lee, Denny, Yavuz, Burak; Diving into Delta Lake-Unpacking the Transaction Log; 03.2020; Databricks

[5] Lee, Denny, Neumann, Andreas; Diving into Delta Lake-Enforcing and Evolving the Schema; 04.2020; Databricks

[6] Lee, Denny, Das, Tarthagarta; Diving into Delta Lake-How do DELETE, UPDATE, and MERGE work; 04.2020; Databricks

[7] WIESE, Lena. Advanced Data Management. De Gruyter, 2015.

[8] MATTHIESSEN, Günter; UNTERSTEIN, Michael. Relationale Datenbanken und Standard-SQL: Konzepte der Entwicklung und Anwendung. Pearson Deutschland GmbH, 2008.

[9] Xiaolon, Wang, Databricks Data Insight Open Course - An Introduction to Delta Lake, 2022

[10] kaggle Data scientist survey , 2020;https://www.kaggle.com/kaggle-survey-2020

[11] Fivetran data analysts a critical underutilized resources, 2020, https://get.fivetran.com/state-of-data-industry-results.html?$_g l = 1 * vbe84x *_g a * MTgxMTI4NDgwMy4xNjgwNDgxMTUz *_g a_N E72Z5F3GB * MTY4MDQ4MTE1My4xLjEuMTY4MDQ4MTMwMC41OC4wLjA.TheGuideTo

[12] Datensatz: https://www.offenedaten.frankfurt.de/dataset/?groups=soci

[13] https://daten.berlin.de/datensaetze/ausleihen-%C3%B6ffentlichen-bibliotheken-pankow-2022