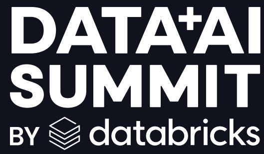


Best Features of Delta Lake: Love your open tables

Youssef Mrini : Senior Solution Engineer at Databricks
Matthew Powers : Staff Developer Advocate at Databricks

©2024 Databricks Inc. — All rights reserved

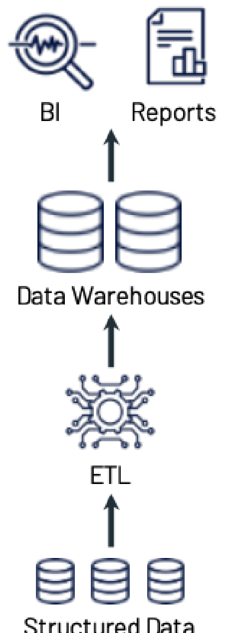
Agenda

The awesome features we will cover today

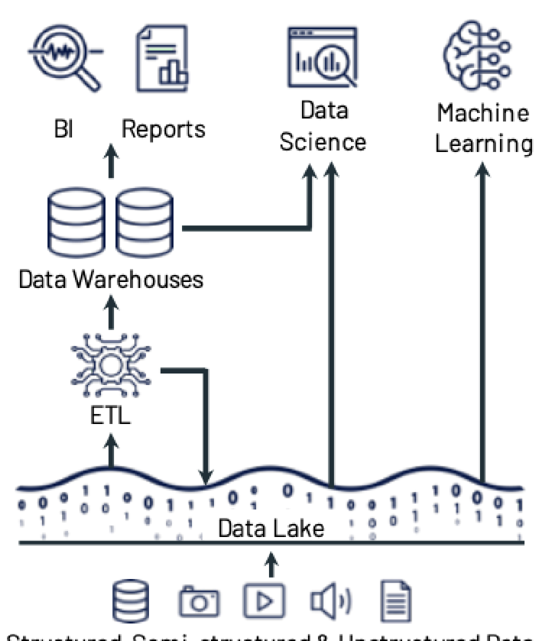
- Data intelligence architecture
- Reliable transactions
- Rollback
- Compatible with pandas/Polars/DataFusion/PySpark
- Safe partition operations
- Delete rows (boosted by deletion vectors)
- Powerful merge operations
- Schema evolution
- Generated columns
- Constraints and checks
- Liquid clustering

©2024 Databricks Inc. — All rights reserved

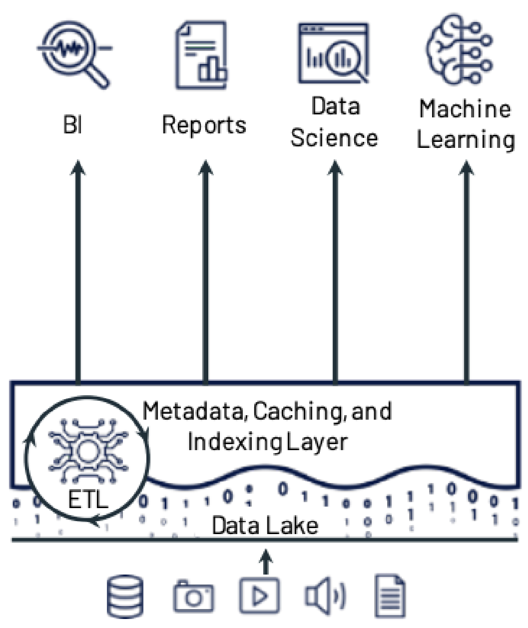
Lakehouse architecture



(a) First-generation platforms.



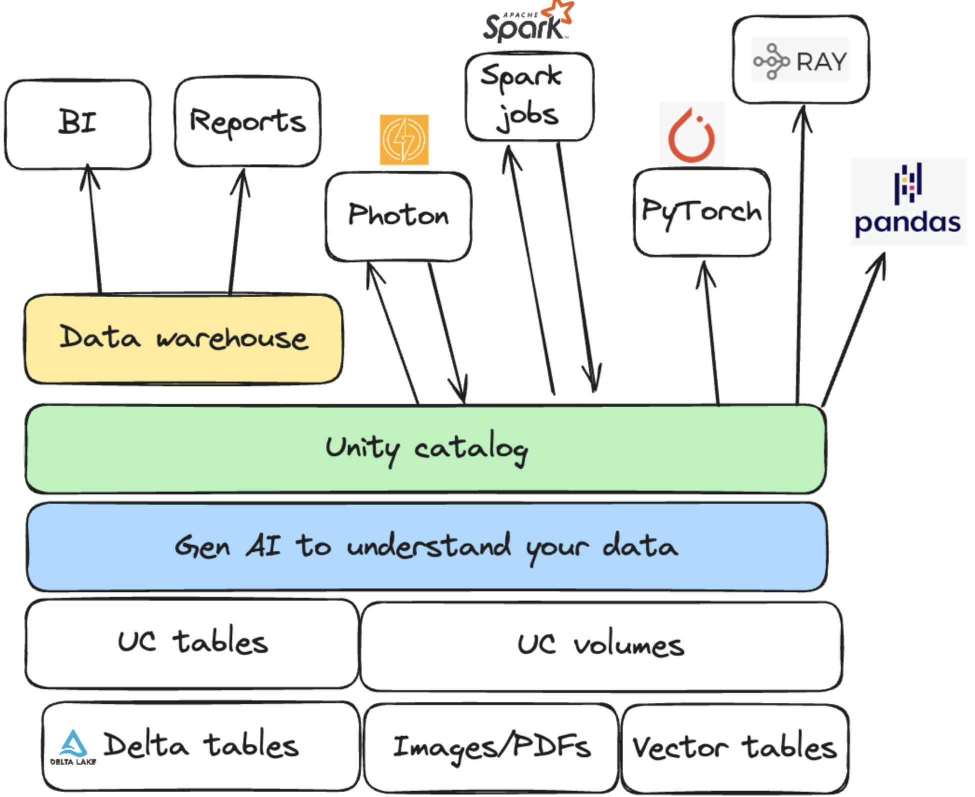
(b) Current two-tier architectures.



(c) Lakehouse platforms.

©2024 Databricks Inc. — All rights reserved

Data intelligence architecture



©2024 Databricks Inc. — All rights reserved

Reliable transactions

1: Appending to a Parquet data lake is dangerous

```
data2 = [("annita", "brasil")]
rdd2 = spark.sparkContext.parallelize(data2)
df2 = rdd2.toDF(columns)

df2.repartition(1).write.mode("append").format("parquet").save("tmp/singers1")
```

no guarantee transaction will complete
no concurrency protection
no schema enforcement or column constraints

2: Appending to a Delta table is safe


```
df2.repartition(1).write.mode("append").format("delta").save("tmp/singers2")
```

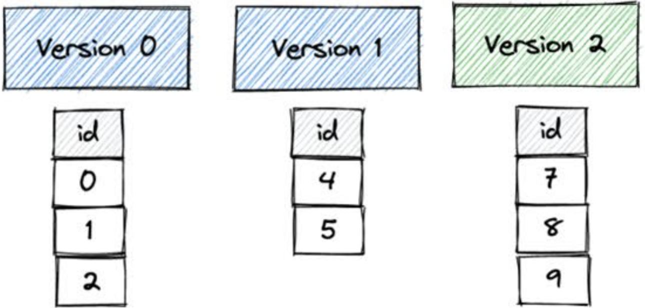
* atomic
* consistent
* isolated
* durable

Examples of Transactions:

* delete rows
* append
* upsert
* overwrite rows
* compact small files

©2024 Databricks Inc. — All rights reserved

Rollback with RESTORE



2: Rollback to version 1

```
deltaTable = DeltaTable.forPath(spark, "/tmp/delta-table")
deltaTable.restoreToVersion(1)
```

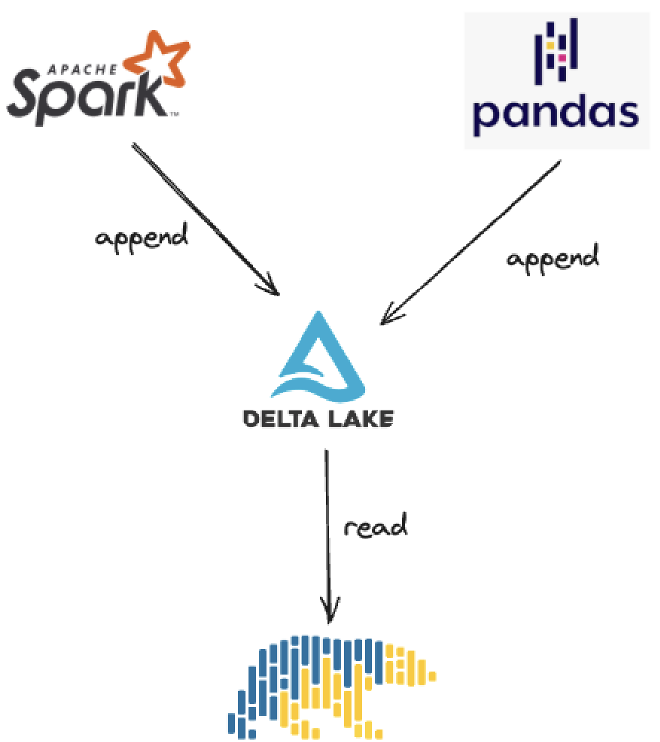
3: Confirm that the correct data is read

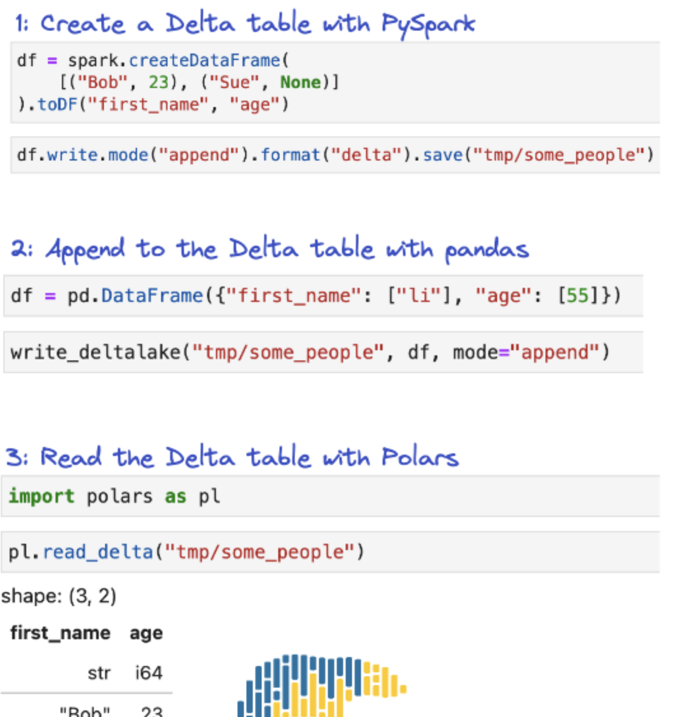
```
spark.read.format("delta").load("/tmp/delta-table").show()

+---+
| id|
+---+
|  4|
|  5|
+---+
```

data as of Version 1

©2024 Databricks Inc. — All rights reserved

Compatible with pandas/polars/Pyspark





©2024 Databricks Inc. — All rights reserved

Safe partition operations

1: Delta table partitioned by country

```
spark-warehouse/country_people
├── _delta_log
│   └── 00000000000000000000.json
├── country=Argentina
│   └── part-00000-03ceafc8-b9b5-4309-8457-6e50814aaa8b.c000.snappy.parquet
├── country=China
│   └── part-00000-9a8d67fa-c23d-41a4-b570-a45405f9ad78.c000.snappy.parquet
└── country=Russia
    └── part-00000-c49ca623-ea69-4088-8d85-c7c2de30cc28.c000.snappy.parquet
```

2: Add a partition to the table

```
df.repartition(F.col("country")).write.mode("append").partitionBy("country").format(
    "delta"
).saveAsTable("country_people")
```

better than Hive ADD PARTITION

3: Remove a partition from the table

```
dt = delta.DeltaTable.forName(spark, "country_people")

dt.delete(F.col("country") == "Argentina")
```

better than Hive REMOVE PARTITION

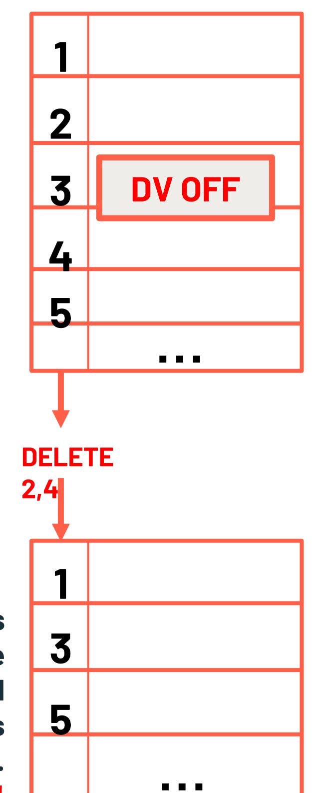
©2024 Databricks Inc. — All rights reserved

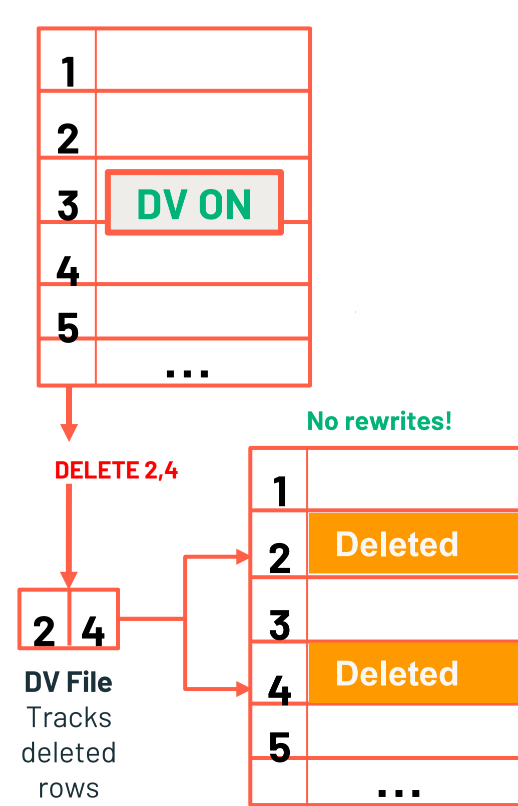
Delete rows

How does it Work ?

- Deletion Vectors remove the need to re-write files when rows in files are deleted. Instead, the feature allows Delta to track the deleted rows in a bitmap file during DELETE, MERGE and UPDATE.

- **Deletion Vectors** provide up to **10x performance** improvements for DELETE, MERGE and UPDATE

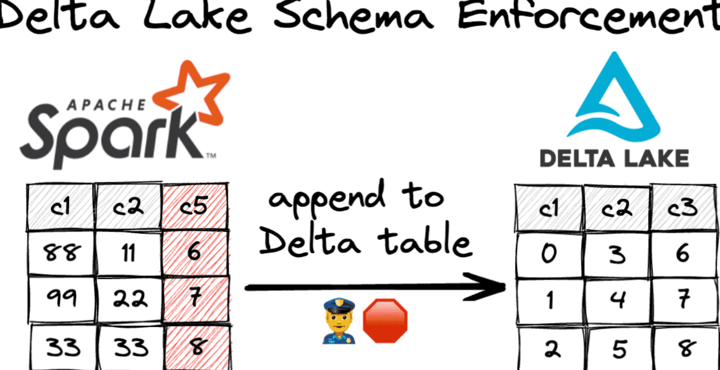




©2024 Databricks Inc. — All rights reserved

Schema enforcement

Schema enforcement is a Delta Lake feature that prevents you from appending data with a different schema to a table



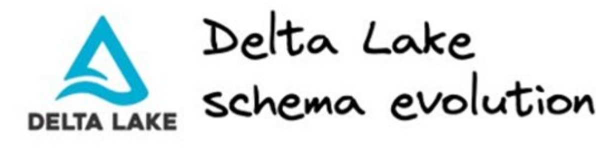

```
AnalysisException: A schema mismatch detected when writing to the Delta table (Table ID: c4ac94bc-32d1-4a83-96fd-7c2
4cf5f6b).
To enable schema migration using DataFrameWriter or DataStreamWriter, please set:
'.option("mergeSchema", "true")'.
```

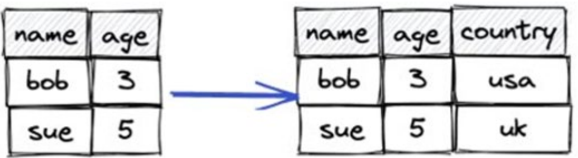

©2024 Databricks Inc. — All rights reserved

Schema evolution

Schema evolution in Delta Lake is a flexible capability that allows for the modification of a table's schema to accommodate changing requirements. This feature enables the addition of new columns, modification of column types, renaming columns, or deleting columns

- Delta table with 2 columns : name, age
- Dataframe has 3 columns: name, age and country
- The requirement is to write this data to the same table





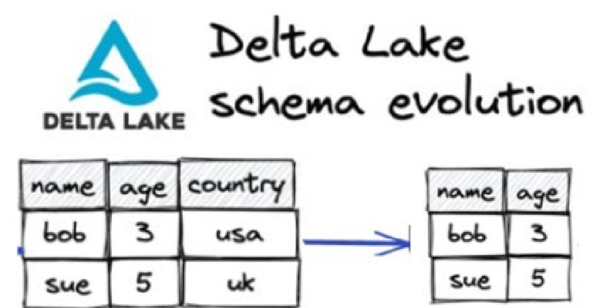

```
df.write.mode('append').option('mergeSchema', 'true').saveAsTable('personal_info')
```


©2024 Databricks Inc. — All rights reserved

Schema evolution

Schema evolution in Delta Lake is a flexible capability that allows for the modification of a table's schema to accommodate changing requirements. This feature enables the addition of new columns, modification of column types, renaming columns, or deleting columns

- Delta table with 3 columns : name, age ,country
- Dataframe has 2 columns: name and age
- The requirement is to write this data to the same table and overwrite the data and schema of the existing Delta table



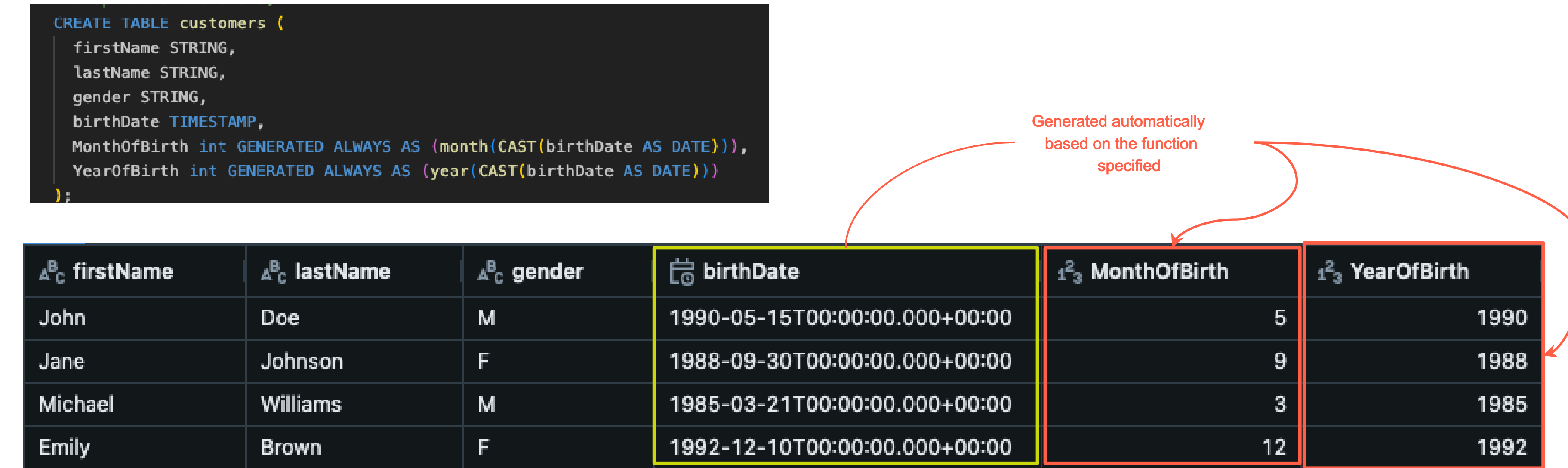

```
df.write.mode('overwrite').option('overwriteSchema', 'true').saveAsTable('personal_info')
```


©2024 Databricks Inc. — All rights reserved

Generated columns

Delta Lake supports generated columns which are a special type of column whose values are automatically generated based on a user-specified function over other columns in the Delta table.

- `GENERATED BY DEFAULT`: insert operations can specify values for the identity column.
- `GENERATED ALWAYS` : override the ability to manually set values.



```
CREATE TABLE customers (
  firstName STRING,
  lastName STRING,
  gender STRING,
  birthDate TIMESTAMP,
  MonthOfBirth int GENERATED ALWAYS AS (month(CAST(birthDate AS DATE))),
  YearOfBirth int GENERATED ALWAYS AS (year(CAST(birthDate AS DATE)))
);
```

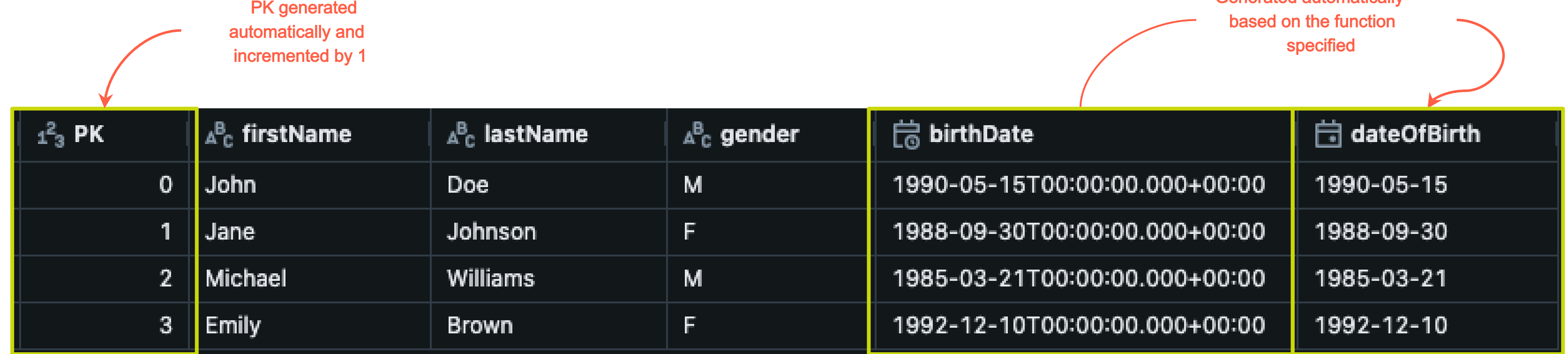
firstName	lastName	gender	birthDate	MonthOfBirth	YearOfBirth
John	Doe	M	1990-05-15T00:00:00.000+00:00	5	1990
Jane	Johnson	F	1988-09-30T00:00:00.000+00:00	9	1988
Michael	Williams	M	1985-03-21T00:00:00.000+00:00	3	1985
Emily	Brown	F	1992-12-10T00:00:00.000+00:00	12	1992

©2024 Databricks Inc. — All rights reserved

Identity Columns

Delta Lake identity columns are a type of generated column that assigns unique values for each record inserted into a table. The following example shows the basic syntax for declaring an identity column during a create table statement

```sql
CREATE TABLE customers_identity (
  PK BIGINT GENERATED ALWAYS AS IDENTITY (start with 0 INCREMENT by 1),
  firstName STRING,
  lastName STRING,
  gender STRING,
  birthDate TIMESTAMP,
  dateOfBirth DATE GENERATED ALWAYS AS (CAST(birthDate AS DATE))
);
```



PK	firstName	lastName	gender	birthDate	dateOfBirth
0	John	Doe	M	1990-05-15T00:00:00.000+00:00	1990-05-15
1	Jane	Johnson	F	1988-09-30T00:00:00.000+00:00	1988-09-30
2	Michael	Williams	M	1985-03-21T00:00:00.000+00:00	1985-03-21
3	Emily	Brown	F	1992-12-10T00:00:00.000+00:00	1992-12-10

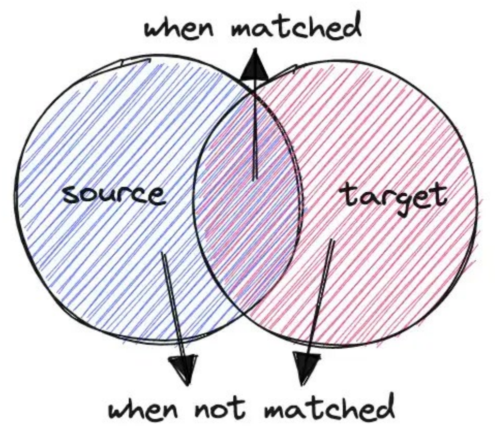
©2024 Databricks Inc. — All rights reserved

Merge operation

Delta Lake supports inserts, updates, and deletes in `MERGE`, and it supports extended syntax beyond the SQL standards to facilitate advanced use cases.

To reduce the time taken by merge reduce the search space for matches.

```
MERGE INTO target
USING source
ON source.key = target.key
WHEN MATCHED THEN
  UPDATE SET target.lastSeen = source.timestamp
WHEN NOT MATCHED THEN
  INSERT (key, lastSeen, status) VALUES (source.key,  source.timestamp, 'active')
WHEN NOT MATCHED BY SOURCE AND target.lastSeen >= (current_date() - INTERVAL '5' DAY) THEN
  UPDATE SET target.status = 'inactive'
```



©2024 Databricks Inc. — All rights reserved

Constraints and checks

Delta Lake supports the implementation of constraints to ensure data integrity and quality within Delta tables.
These constraints are used to enforce rules on the data being inserted or updated in the table, preventing bad data from being added. There are two main types of constraints supported by Delta Lake:

NOT NULL Constraint

```
CREATE TABLE events (
id LONG NOT NULL,
  date STRING NOT NULL,
  location STRING,
  description STRING);

--Adding a Not Null constraint
ALTER TABLE events CHANGE COLUMN date DROP NOT NULL;
--Dropping a Not Null constraint
ALTER TABLE events CHANGE COLUMN id   SET  NOT NULL;
```

Key Features

- Constraints are set at table schema level
- You can create or drop NOT NULL constraints using the ALTER TABLE CHANGE COLUMN command

©2024 Databricks Inc. — All rights reserved

Constraints and checks

Delta Lake supports the implementation of constraints to ensure data integrity and quality within Delta tables.
These constraints are used to enforce rules on the data being inserted or updated in the table, preventing bad data from being added. There are two main types of constraints supported by Delta Lake:

CHECKS

```
CREATE TABLE events (
id LONG NOT NULL,
  date STRING NOT NULL,
  location STRING,
  description STRING);

--Adding a check constraint
ALTER TABLE events ADD CONSTRAINT  dateWithinRange CHECK (date > '1900-01-01');
--Dropping a check constraint
ALTER TABLE events DROP CONSTRAINT dateWithinRange;
```

Key Features

- ALTER TABLE ADD CONSTRAINT verifies that all existing rows satisfy the constraint before adding it to the table

©2024 Databricks Inc. — All rights reserved

Liquid clustering

Introduces a new incremental clustering approach

Delta Lake liquid clustering replaces table partitioning and ZORDER to simplify data layout decisions and optimize query performance. It provides flexibility to redefine clustering keys without rewriting existing data, allowing data layout to evolve alongside analytic needs over time.

```
-- Create a table customers_liquid clustered by gender
CREATE TABLE customers_liquid (
  firstName STRING, lastName STRING, gender STRING, birthDate TIMESTAMP, salary INT
) cluster by (gender);

-- Create a table Using CTAS statement
CREATE TABLE customers_liquid cluster by (gender) as select * from customers;

--Optimize/ Cluster the table without specifiying the columns
optimize customers_liquid;
```


©2024 Databricks Inc. — All rights reserved

Liquid clustering

Introduces a new incremental clustering approach

Which columns to choose when clustering a table ?

- Tables often filtered by high cardinality columns.
- Tables with significant skew in data distribution.
- Tables that grow quickly and require maintenance and tuning effort.
- Tables with concurrent write requirements.
- Tables with access patterns that change over time.
- Tables where a typical partition key could leave the table with too many or too few partitions.

©2024 Databricks Inc. — All rights reserved

DATA+AI SUMMIT

Thank you
Any questions ?

©2024 Databricks Inc. — All rights reserved