Implementing Data Quality: A Practical Guide on Data Quality Expectations with Delta Live Tables

Matthew Norberg

Session Overview

- Introduction
- Part I: Defining Data Quality & Data Expectations
  - Define data quality and data expectations.
  - Learn my four-step process to write data expectations.
  - Introduce demo data we will be using for part II.
- Part II: Data Expectations Demo
  - Short demo on creating data quality expectations
  - Lessons learned and things you will encounter when you do this yourself.
- Conclusion

Why do we care about data quality?

- Trust reporting results by providing accurate analytics, leading to better decision-making
- Essential for building accurate AI and ML models which rely on clean quality data
- Poor data quality is costly!

  Gartner: Every year, poor data quality costs organizations an average of $12.9 million (2021).

Sources:

- Gartner. "12 Actions to Improve Your Data Quality." Gartner, 14 July 2021. https://www.gartner.com/smarterwithgartner/how-to-improve-your-data-quality.
- Databricks. "Data Quality: Ensure Accuracy for Better Decisions." Databricks, 2023. https://www.databricks.com/glossary/data-quality.

“You can have all the AI in the world, but if it's on a shaky data foundation, then it's not going to bring you any value.”

Carol Clements, chief digital and technology officer, JetBlue

Source:
Terry, Matt. "Unlocking Enterprise AI." *Economist Impact*, November 22, 2024. https://impact.economist.com/perspectives/technology-innovation/unlocking-enterprise-ai.

Part I: Defining Data Quality & Data Expectations

What is Data Quality?

Data quality measures how well a dataset meets criteria for *accuracy, completeness, validity, consistency, uniqueness, timeliness and fitness for purpose*, and it is critical to all data governance initiatives within an organization.

Source: IBM

Source:
IBM. "What Is Data Quality?" *IBM*, 2021. https://www.ibm.com/topics/data-quality.

Dimensions of Data Quality

Before

Name	Gender	Street	House #	Zip code	City	State	D.O.B
John Doe	Male	60th street	45		New York	New York	08/12/64
Jane Doe	Male	Jonathan ln	36	10023	Poughkeepsy	NY	21-dec-1954

After

Name	Gender	Street	House #	Zip code	City	State	D.O.B
John Doe	Male	**E 60th St**	45W	**10022**	New York	**NY**	08/12/64
Jane Doe	**Female**	**Jonathan Lane**	36	10023	**Poughkeepsie**	NY	**12/21/54**

Completeness Accuracy Conformity Consistency Uniqueness

Before

Name	Address	Postal Code	City	State
John Smith	545 S Valley View Drive # 136	34563	Anytown	New York
Margaret & John smith	545 Valley View ave unit 136	34563-2341	Anytown	New York
Maggie Smith	545 S Valley View Dr		Anytown	New York
John Smith	545 Valley Drive St.	34253	NY	NY

After

Name	Address	Zip Code	City	State	Cluster
John Smith	545 S Valley View Drive # 136	34563	Anytown	New York	1
Margaret & John smith	545 Valley View ave unit 136	34563-2341	Anytown	New York	1
Maggie Smith	545 S Valley View Dr		Anytown	New York	1
John Smith	545 Valley Drive St.	34253	NY	NY	2

Source:
Gschwind, Mark. "Enterprise Information Management (EIM) in SQL Server 2012." SlideShare. February 25, 2013. https://www.slideshare.net/slideshow/gschwind-mdsdqs-sqlsatmv/16761590#4.

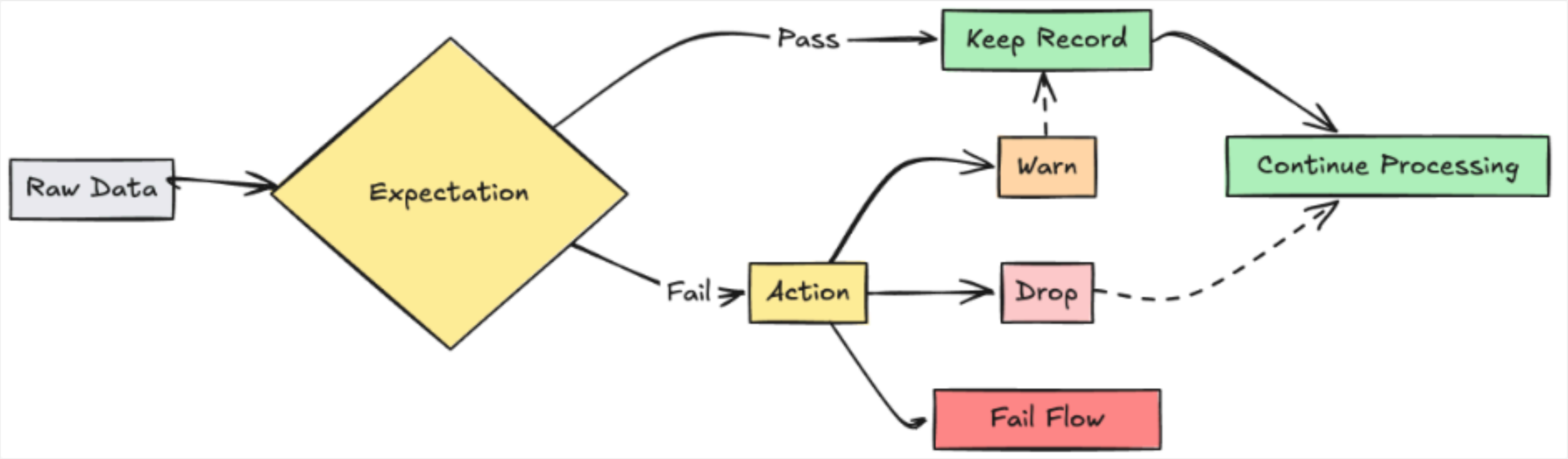
What are Data Expectations?

- A rule that specifies what your data should look like.
- Used to test your data against these rules in a Delta Live Table (DLT) pipeline.
- Typically placed between bronze and silver layers of the lakehouse.

Why Write Data Expectations?

- Allows you to measure proportion of data that meets quality standards.
- Quarantine bad data to prevent it from entering downstream processes.
- Identify bugs in code that cause bad data.
- First step towards data quality reporting with Lakehouse Monitoring or other BI software.

Conceptual Diagram



Source:
Databricks. "Manage Data Quality with Pipeline Expectations." Databricks Documentation. Last modified January 17, 2025. Accessed January 22, 2025. https://docs.databricks.com/en/delta-live-tables/expectations.html#manage-data-quality-with-pipeline-expectations.

NYC Taxi Data

Just now (3s) 9 SQL

```
%sql
select * from trips
```

See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [tpep_pickup_datetime: timestamp, tpep_dropoff_datetime: timestamp ... 4 more fields]

Table +

	tpep_pickup_datetime	tpep_dropoff_datetime	1.2 trip_distance	1.2 fare_amount	123 pickup_zip	123 dropoff_zip
1	2016-02-13T21:47:53.000+00:00	2016-02-13T21:57:15.000+00:00	1.4	8	10103	10110
2	2016-02-13T18:29:09.000+00:00	2016-02-13T18:37:23.000+00:00	1.31	7.5	10023	10023
3	2016-02-06T19:40:58.000+00:00	2016-02-06T19:52:32.000+00:00	1.8	9.5	10001	10018
4	2016-02-12T19:06:43.000+00:00	2016-02-12T19:20:54.000+00:00	2.3	11.5	10044	10111
5	2016-02-23T10:27:56.000+00:00	2016-02-23T10:58:33.000+00:00	2.6	18.5	10199	10022

NYC Taxi Data Pretty Good But ....

1.2 trip_distance	1.2 fare_amount
0.16	-8
0.48	-4.5
0.52	-4
25	0
0	0
26.3	0
4.1	0
17.1	0.01
4.9	0.01
0	0.01

123 pickup...	123 dropoff_zip
7002	7002
7087	11231
7114	7114
7311	7311
7737	7758

	tpep_pickup_datetime	tpep_dropoff_datetime	1.2 trip_distance	1.2 fare_amount	123 pickup_zip	123 dropoff_zip
84	2016-01-07T12:53:17.000+00:00	2016-01-07T12:53:53.000+00:00	0	52	11422	11422

Terrible NYC Taxi Data Set Creation

- Modified NYC Taxi Data to purposely create data quality issues using User Defined Functions (UDFs) and data transformation techniques.
- Added `driver` column containing name of taxi driver.
- Created driver information table with age, years driving, state of origin, and phone number.

Terrible Trips Table

Just now (3s)

10

SQL

```
%sql
select * from terrible_trips
```

See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [tpep_pickup_datetime: timestamp, tpep_dropoff_datetime: timestamp ... 5 more fields]

Table

	tpep_pickup_datetime	tpep_dropoff_datetime	trip_distance	fare_amount	pickup_zip	dropoff_zip	driver
1	2016-02-13T21:47:53.000+00:00	2016-02-13T21:57:15.000+00:00	3	10.3	86132	10110	Alison Acosta
2	2016-02-13T18:29:09.000+00:00	2016-02-13T18:37:23.000+00:00	19	69.87	92265	10023	Lorraine Marsh
3	2016-02-06T19:40:58.000+00:00	2016-02-06T19:52:32.000+00:00	2	1	84406292	10018	Antonio Ferguson
4	2016-02-12T19:06:43.000+00:00	2016-02-12T19:20:54.000+00:00	15	47.17	79932	10111	Melanie Reed
5	2016-02-23T10:27:56.000+00:00	2016-02-23T10:58:33.000+00:00	15	11	70641	10022	Anne Lane

Taxi Drivers Table

Just now (3s)

11

```
%sql
select * from taxi_drivers
```

See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [driver: string, age: integer ... 3 more fields]

Table

	driver	age	years_driving	state	phone_number
1	Stephanie Hensley	34	30	Montana	(456)626-6413x0113
2	Brian Wong	97	25	null	611-295-2665x21465
3	Monica Burke	73	25	NM	(527)522-8398x5955
4	Karen Cross	140	45	CO	+1-808-523-7557x6563
5	Jonathan Conley	114	12	New Hampshire	(994)428-5333

Data Expectation Examples

```sql
CREATE OR REFRESH MATERIALIZED VIEW Terrible_Trips_DQ_SQL(
  CONSTRAINT distance_example EXPECT (trip_distance > 0)
)
AS SELECT * FROM `dev-sandbox-catalog`.dais.terrible_trips
```

```python
trip_rules = {}
driver_rules = {}

driver_rules["age_example"] = "age > 0"

trip_rules["distance_example"] = "trip_distance > 0"

trip_rules["driver_example"] = "driver is not null"
```

Expectation Vs SQL Relationship

```
%sql
select * from taxi_drivers
where age > 0
```

See performance (1)

Original Data Expectation:

```
driver_rules["age_example"] = "age > 0"
```

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [driver: string, age: integer ... 3 more fields]

Table

	driver	age	years_driving	state	phone_number
1	Stephanie Hensley	34	30	Montana	(456)626-6413x0113
2	Brian Wong	97	25	null	611-295-2665x21465
3	Monica Burke	73	25	NM	(527)522-8398x5955
4	Karen Cross	140	45	CO	+1-808-523-7557x6563
5	Jonathan Conley	114	12	New Hampshire	(994)428-5333

A Note On The Code

- Write expectations as specifications for how your data ***should*** look.
    - This is not how we usually think about data quality issues.

3 minutes ago (14s)

15

```sql
select * from taxi_drivers
where age <= 0
```

See performance (1)

Expectation:

```python
driver_rules["age_example"] = "age > 0"
```

Table

	driver	age	years_driving	state	phone_number
1	Stephanie Rice	0	16	West Virginia	+1-898-401-7359x80265
2	Emily Hall	0	24	Oklahoma	(992)581-1574
3	Jenny Wells	0	21	New Jersey	001-261-593-0609x74865

Steps to write good expectations

1. Profile your data
2. Talk to business owners and domain experts
3. Document and publish data rules
4. Transform rules to code

Step 1: Profile Your Data

- Profiling data doesn’t have to be complex.
- You can usually find data quality issues with simple `SELECT  *` queries.
- Ask: What are the outliers in each column?
- Tips:
  - Sort data in both ascending and descending order.
  - Use SQL to search for missing values.
  - Keep data quality dimensions in mind throughout the process.

Step 2: Converse With Data Owners

- Understanding business context will help you define data quality expectations.
- **Example:**
  - You are analyzing weather data in San Francisco, and you want to set a lower bound on the temperature.
  - An expert could tell you that temperatures are unlikely to fall below 20°F, given the record low of 27°F on December 11, 1932.

Source:
Extreme Weather Watch. "Lowest Temperatures in San Francisco History." Accessed January 28, 2025. https://www.extremeweatherwatch.com/cities/san-francisco/lowest-temperatures.

Step 3: Write Down Your Rules

- Perform this step alongside Steps 1 and 2.
- As you profile your data and consult with domain experts, maintain a list of identified data quality issues.
- By documenting each issue during the first two steps, you will have completed Step 3 immediately after finishing step 2.

Step 4: Transform Rules To Code

- Transform each item in the list from the previous step into SQL clauses.
- Basically, anything that you could place after a `WHERE` statement in a SQL query is a valid expectation.
- **Example:**
  - Suppose your list contains the entry ‘age should be positive and less than 125’.
  - This would translate to `age > 0 AND age < 125`.

Part II: Demo

Step 1: Profile Taxi Data

Table

	tpep_pickup_datetime	tpep_dropoff_datetime	trip_distance	fare_amount	pickup_zip	dropoff_zip	driver
1	2016-02-13T21:47:53.000+00:00	2016-02-13T21:57:15.000+00:00	3	10.3	86132	10110	Alison Acosta
2	2016-02-13T18:29:09.000+00:00	2016-02-13T18:37:23.000+00:00	19	69.87	92265	10023	Lorraine Marsh
3	2016-02-06T19:40:58.000+00:00	2016-02-06T19:52:32.000+00:00	2	1	84406292	10018	Antonio Ferguson
4	2016-02-12T19:06:43.000+00:00	2016-02-12T19:20:54.000+00:00	15	47.17	79932	10111	Melanie Reed
5	2016-02-23T10:27:56.000+00:00	2016-02-23T10:58:33.000+00:00	15	11	70641	10022	Anne Lane

Table

	driver	age	years_driving	state	phone_number
1	Stephanie Hensley	34	30	Montana	(456)626-6413x0113
2	Brian Wong	97	25	null	611-295-2665x21465
3	Monica Burke	73	25	NM	(527)522-8398x5955
4	Karen Cross	140	45	CO	+1-808-523-7557x6563
5	Jonathan Conley	114	12	New Hampshire	(994)428-5333

Tip 1: Order Your Data

trip_distance	fare_amount	pickup_zip	dropoff_zip	driver
-99	-1.4	78107	10020	Cameron Thomas
-99	6.2	44664	10018	Craig Butler
-99	11.16	64335	10030	Nicole Martin
-99	79.4	82697436	10153	Jared Dixon
-98	73.77173	553	10011	Jason Williams
-98	10000010	30798	10021	Katherine Jones
-98	66.4	58754	10111	Devin Thompson
-98	79.3	78296	10025	Duplicate Taxi Driver
-97	62	50840	10017	William Baxter
-97	82	86429737	10014	David Spencer

trip_distance	fare_amount	pickup_zip	dropoff_zip	driver
9976746	61.9	42760	10103	Melanie Jensen
9957722	41.75	51293	10021	Laurie Clarke
9954053	50	16235	10019	Diana Harding
9938728	0.5	90334	10044	Phillip Reyes
9926747	89.56	50969	11211	Robin Meyer
9861686	20.8	10235	10044	Autumn Conner
9849742	49	87912	10023	Christopher Mcpherson
9830614	56.9	64474	11225	Jordan Schmitt
9819629	10000368.7	92704	10012	Lonnie Schaefer
9812208	15	45469	10119	Brandy Jensen

Tip 2: Look For Missing Values

Table

driver is null | Add filter

	tpep_pickup_datetime	tpep_dropoff_datetime	trip_distance	fare_amount	pickup_zip	dropoff_zip	driver
1	2016-02-20T22:09:04.000+00:00	2016-02-20T22:22:43.000+00:00	9	35.44	64714	10003	null
2	2016-02-17T15:54:42.000+00:00	2016-02-17T16:04:41.000+00:00	17	50.3	57000	10021	null
3	2016-02-19T07:47:10.000+00:00	2016-02-19T07:57:32.000+00:00	13	36.74892	47424	10171	null
4	2016-02-23T06:56:26.000+00:00	2016-02-23T07:00:30.000+00:00	7	54.2	57759	10199	null
5	2016-02-16T16:18:30.000+00:00	2016-02-16T16:25:13.000+00:00	3	10000278	17028	10021	null
6	2016-02-02T09:45:45.000+00:00	2016-02-02T10:02:27.000+00:00	1	48.36	42491	10111	null
7	2016-02-09T08:11:51.000+00:00	2016-02-09T08:19:24.000+00:00	0	9.6	62563	10028	null
8	2016-02-17T21:28:58.000+00:00	2016-02-17T21:32:40.000+00:00	10	10000454.4	10227	10011	null
9	2016-02-24T12:57:31.000+00:00	2016-02-24T13:05:01.000+00:00	18	95.2	59420	10011	null
10	2016-02-06T12:19:53.000+00:00	2016-02-06T12:24:22.000+00:00	11	18.55	25907	10065	null

Tip 3: Does This Make Sense?

Table

	driver	age	years_driving	state	phone_number
39	Mary Mendoza	[illegible]	49	North Dakota	[illegible]
40	Joseph Lee	68	49	Louisiana	001-207-579-6798x74440
41	Jamie Romero	48	49	OH	+1-961-406-3911x71575
42	John Harrison	39	49	Iowa	890.368.0225
43	Terri Becker	9	49	Kentucky	001-777-708-5336x969
44	Christopher Fu...	7	49	Illinois	(536)404-2553
45	Alejandra Ferguson	64	49	MN	711-514-2119x36945
46	Jeffrey Nelson	72	49	VT	001-253-755-8551x61212
47	Angela Bryant	148	49	FL	+1-820-858-0837x584
48	Courtney Valentine	76	49	KY	001-577-971-9884
49	Mrs. Haley Scott	117	49	VA	6404424471

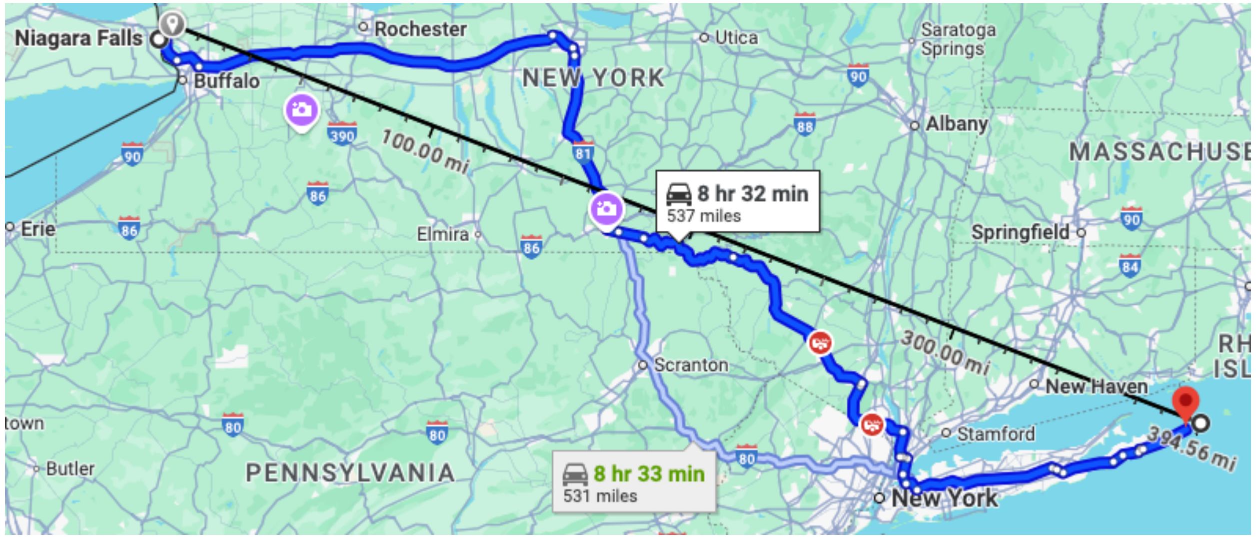
Step 2: Domain Expertise

- **Driver Age**: In NYC, you must be at least 19 years old to obtain a Taxi and Limousine Commission (TLC) license. Therefore, we expect the driver age column to have values of at least 19.
- **Minimum Fare**: The initial charge for a taxi ride in NY is $3.00. Expect fare amount values to be at least $3.00.
- **Zip Codes:** Zip codes are length 5 and can start with 0.

Source:

- NYC Taxi & Limousine Commission. "Taxi Fare Information." NYC.gov. Accessed January 28, 2025. https://www.nyc.gov/site/tlc/passengers/taxi-fare.page.
- NYC Taxi & Limousine Commission. "Get a TLC Driver's License." NYC.gov. Accessed January 28, 2025. https://www.nyc.gov/site/tlc/drivers/get-a-tlc-drivers-license.page.

NY Max Distance



Source:

- NYC Taxi & Limousine Commission. "Taxi Fare Information." NYC.gov. Accessed January 28, 2025. https://www.nyc.gov/site/tlc/passengers/taxi-fare.page.
- Meier, Eric. "What is the Longest Road Trip Possible in New York State?" Lite 98.7. Accessed January 28, 2025. https://lite987.com/longest-roadtrip-possible-in-new-york/.

- Trip distance should be less than 537 miles.
- In NY, Taxis charge $0.70 per 1/5 mile.
- Estimated max fare: $1,882 (400 miles * $0.70 / 0.2 + $3 base charge).
- Considering surcharges, set the upper limit at $2,000.
- Data points exceeding $2,000 are likely invalid.

Step 3: Write Down Your Rules

- This step is essentially done; rules were identified during data profiling and research.
- Here's how I would write down the rules during previous steps, split into two categories:
  - Basic -> Easy to translate into SQL clauses
  - Complex -> Requires reference tables or data transformations to implement

Basic Rules

- Trip distance greater than 0 and less than 400
- Fare amount value greater than $3 and less than $2000
- Pickup and drop off zip codes are length 5
- Taxi driver field is filled out
- Driver's age can not be greater than the number of years they have been driving
- The drop off time can not occur before the pickup time

Complex Rules

- The State column should represent a valid state in the United States
- Driver values should be unique in driver table

Step 4: Transform Rules

English Rule	SQL Statement
Trip distance between 0 and 400 miles	`trip_distance > 0 AND trip_distance < 400`
Fare amount between \$3 and \$1500	`fare_amount > 3 AND fare_amount < 1500`
Pickup zip codes are length 5	`pickup_zip > 10000 AND pickup_zip < 100000`
Taxi driver field is not null	`driver IS NOT NULL`
Years driving is positive and does not exceed age	`Years_driving < age AND years_driving >= 0`

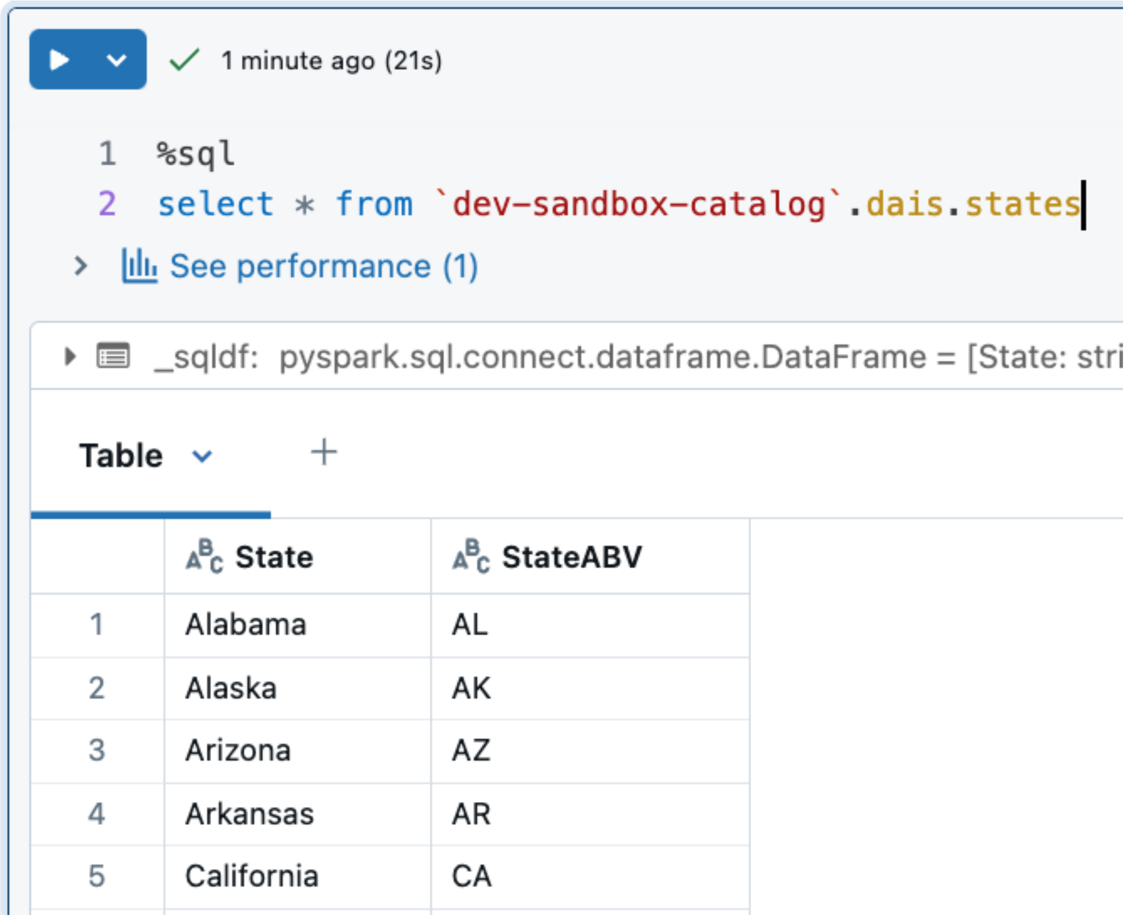
- Not all rules listed in previous slides are shown
- SQL Statement derivations are coming for complex rules !!!

Data Transformation Technique

```python
@dlt.table(
    name="taxi_driver_dq",
    comment="Taxi driver table with data quality expectations applied.",
)
@dlt.expect_all(driver_rules)
def taxi_driver_dq():
    res = spark.sql(
        """
        select
            driver.*,
            num_driver_entries
        from
            `dev-sandbox-catalog`.dais.taxi_drivers driver
        left join (
            select
                driver,
                count(*) as num_driver_entries
            from
                `dev-sandbox-catalog`.dais.taxi_drivers
            group by
                driver
        ) as driver_ct on driver.driver = driver_ct.driver
        """
    )
    return res
```

- Compute the number of times each driver appears in `driver_ct` subquery.
- Join `driver_ct` to `taxi_drivers` to test for uniqueness.
- Expectation: "Drivers should only appear once" translates to `num_driver_entries = 1`.
- Important: Dataframe in function is computed *before* expectations are applied.

Reference Table Techniques



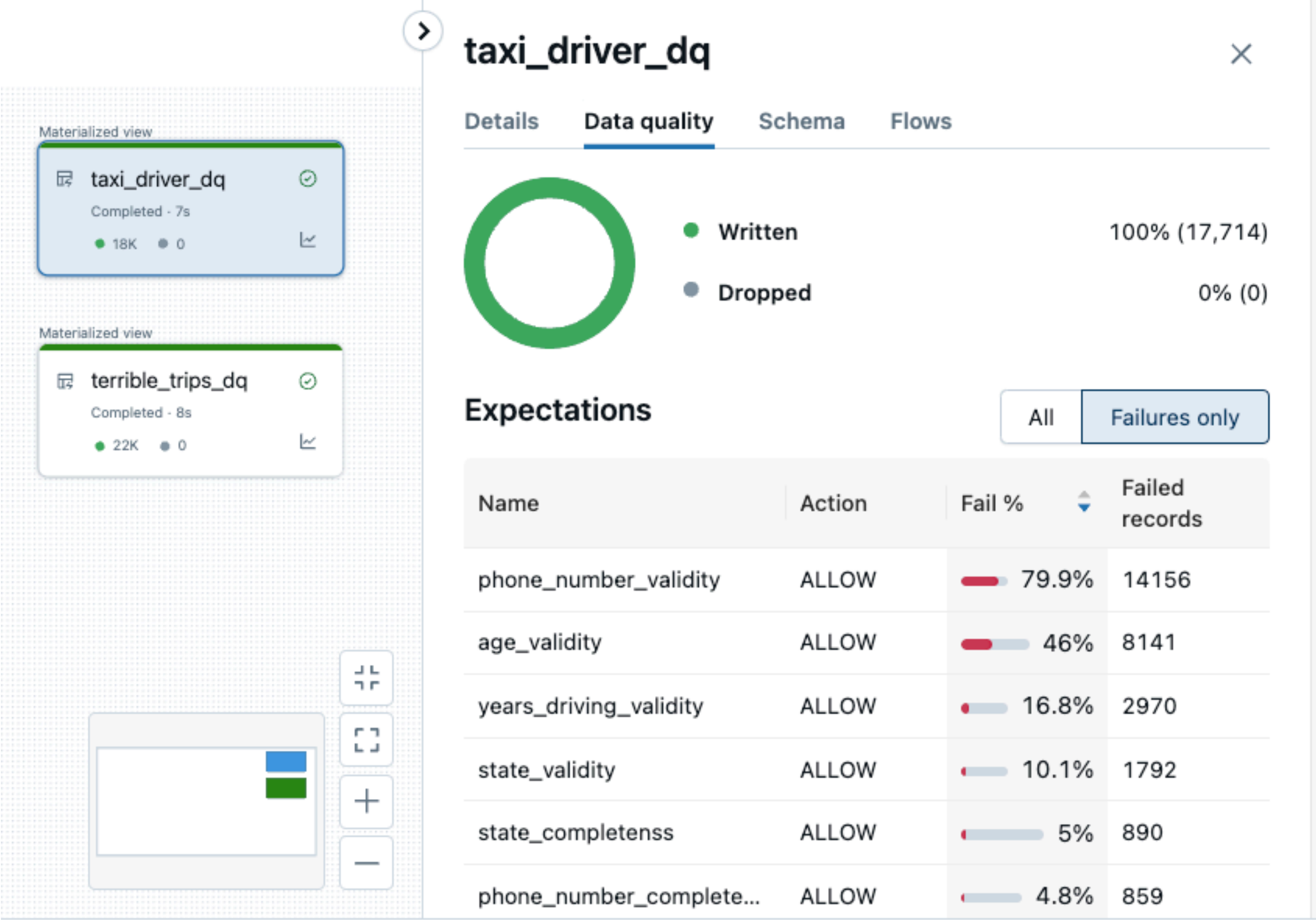
- Similar to the previous data transformation technique.
- Join data to a reference table with valid values.
- A match results in a non-null value in the new column.
- A null value indicates an invalid entry.

Reference Table Techniques II

```python
@dlt.expect_all(driver_rules)
def taxi_driver_dq():
    res = spark.sql(
        """
        select
            driver.*,
            valid_us_state,
            valid_us_state_abv
        from
            `dev-sandbox-catalog`.dais.taxi_drivers driver
        left join (
            select
                state as valid_us_state
            from
                `dev-sandbox-catalog`.dais.states
        ) as vus_states on driver.state = vus_states.valid_us_state
        left join (
            select
                stateabv as valid_us_state_abv
            from
                `dev-sandbox-catalog`.dais.states
        ) as vus_states_abv on driver.state = vus_states_abv.valid_us_state_abv
        """
    )
    return res
```

- If the state is valid and spelled out, `valid_us_state` will not be null.
- If given as an abbreviation, `valid_us_state_abv` will not be null.
- Expectation: `valid_us_state IS NOT NULL OR valid_us_state_abv IS NOT NULL`.
- This is an example of a ”loose” expectation which allows states to be entered in more than one way

The Results



Materialized view
taxi_driver_dq
Completed · 7s
18K
0
Materialized view
terrible_trips_dq
Completed · 8s
22K
0
taxi_driver_dq
Details
Data quality
Schema
Flows
Written
100% (17,714)
Dropped
0% (0)
Expectations
All
Failures only
Name
Action
Fail %
Failed records
phone_number_validity
ALLOW
79.9%
14156
age_validity
ALLOW
46%
8141
years_driving_validity
ALLOW
16.8%
2970
state_validity
ALLOW
10.1%
1792
state_completenss
ALLOW
5%
890
phone_number_complete...
ALLOW
4.8%
859

What Now?

- **View flagged data**: Identify data that failed to meet expectations.
  - **Adjust expectations**: Refine criteria to minimize false positives.
  - **Check unflagged data**: Ensure no bad data slipped through (false negatives).
- **Optionally Quarantine failed data**: Isolate data that doesn't meet standards.
- **Extract results**: Create tables for ongoing monitoring.

Methods To View Expectation Failures

- Recall the SQL Expectation Relationship from earlier
- You could query for the negation of your expectation to find data that failed.
  - `SELECT * FROM taxi_drivers WHERE age < 0`
- Here’s a better way

Add A Passed Column To The Data

7

```
@dlt.table(
    name="taxi_driver_dq_status",
    comment="Will contain additional columns indicating whether a dq c
)
def taxi_driver_dq_status():
    res = dlt.read("taxi_driver_dq")
    for key in sorted(driver_rules.keys()):
        res = res.withColumn(
            key + "_passed_dq_check",
            F.when(F.expr(driver_rules[key]), True).otherwise(False),
        )
    return res
```

col_name	data_type
driver	string
age	int
years_driving	int
state	string
phone_number	string
valid_us_state	string
valid_us_state_abv	string
num_driver_entries	bigint
age_validity_passed_dq_check	boolean
driver_completeness_passed_dq_check	boolean
driver_uniqueness_passed_dq_check	boolean
phone_number_completeness_passed_dq_check	boolean
phone_number_validity_passed_dq_check	boolean

Finding Data That Passed

Just now (3s)

13

```
%sql
-- Find rows that passed state validity
select * from taxi_driver_dq_status where state_validity_passed_dq_check = True
```

See performance (1)

Table

	driver	age	years_driving	state	phone_number	valid_us_state	valid_us_state_abv
1	Stephanie Hensley	34	30	Montana	(456)626-6413x0113	Montana	null
2	Jonathan Conley	114	12	New Hampshi...	(994)428-5333	New Hampshire	null
3	Kyle Anderson	105	49	Missouri	596.275.5847x087	Missouri	null
4	Diane White	101	35	Wyoming	547.437.4234	Wyoming	null
5	Jonathan Potts	53	27	Missouri	null	Missouri	null
6	Alicia Dunn	82	21	North Carolina	833-446-9547x61619	North Carolina	null

Finding Data That Failed

Just now (2s) 13

```
%sql
-- Find rows that passed state validity
select * from taxi_driver_dq_status where state_validity_passed_dq_check = False
```

See performance (1)

Table

	driver	age	years_driving	state	phone_number
26	Raymond Potter	34	39	null	146492297530869
27	Kyle Strickland	60	26	fgxm	(356)301-1956x559
28	Daniel Holland	86	49	hsswg	294.418.9693x9622
29	Kathleen Sanchez	59	30	null	56
30	Emily Fernandez	10	10	kgc	+1-425-801-0593x20034
31	Jessica Taylor	94	8	qrdphefulk	36881745941789533
32	Gina Barry	117	8	ptrfhtti	957-733-8250
33	Anthony Pierce DVM	87	33	null	null

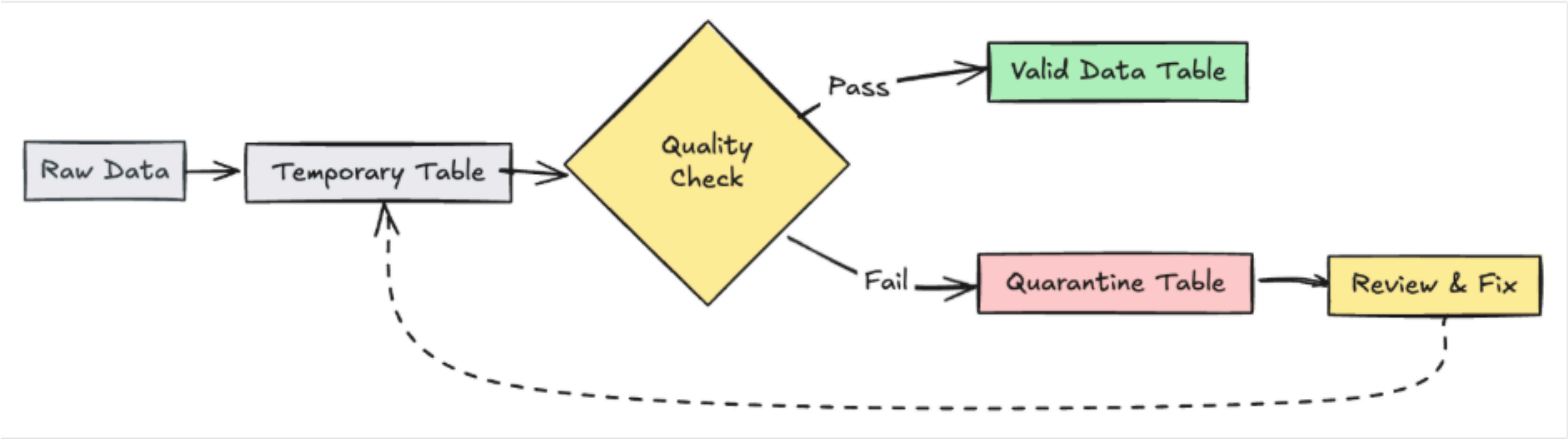
Data Quarantine Procedure

```
# Quarantine Procedure
quarantine_rules = "NOT({0})".format(" AND ".join(driver_rules.values()))

@dlt.table(
    temporary=True
)
def quarantine_drivers():
    res = dlt.read("taxi_driver_dq")
    return res.withColumn("is_quarantined", F.expr(quarantine_rules))

@dlt.table()
def quarantined_drivers():
    res = dlt.read("quarantine_drivers")
    return res.filter("is_quarantined=true")

@dlt.table()
def valid_drivers():
    res = dlt.read("quarantine_drivers")
    return res.filter("is_quarantined=false")
```

Source

Databricks. "Quarantine Invalid Records." Databricks Documentation. Last modified January 17, 2025. Accessed January 22, 2025. https://docs.databricks.com/en/delta-live-tables/expectation-patterns.html#quarantine-invalid-records.

Data Quarantine Procedure Visualized



Source
Databricks. "Quarantine Invalid Records." Databricks Documentation. Last modified January 17, 2025. Accessed January 22, 2025. https://docs.databricks.com/en/delta-live-tables/expectation-patterns.html#quarantine-invalid-records.

Data Quarantine Results

```
%sql
select * from valid_drivers
```

See performance (1)

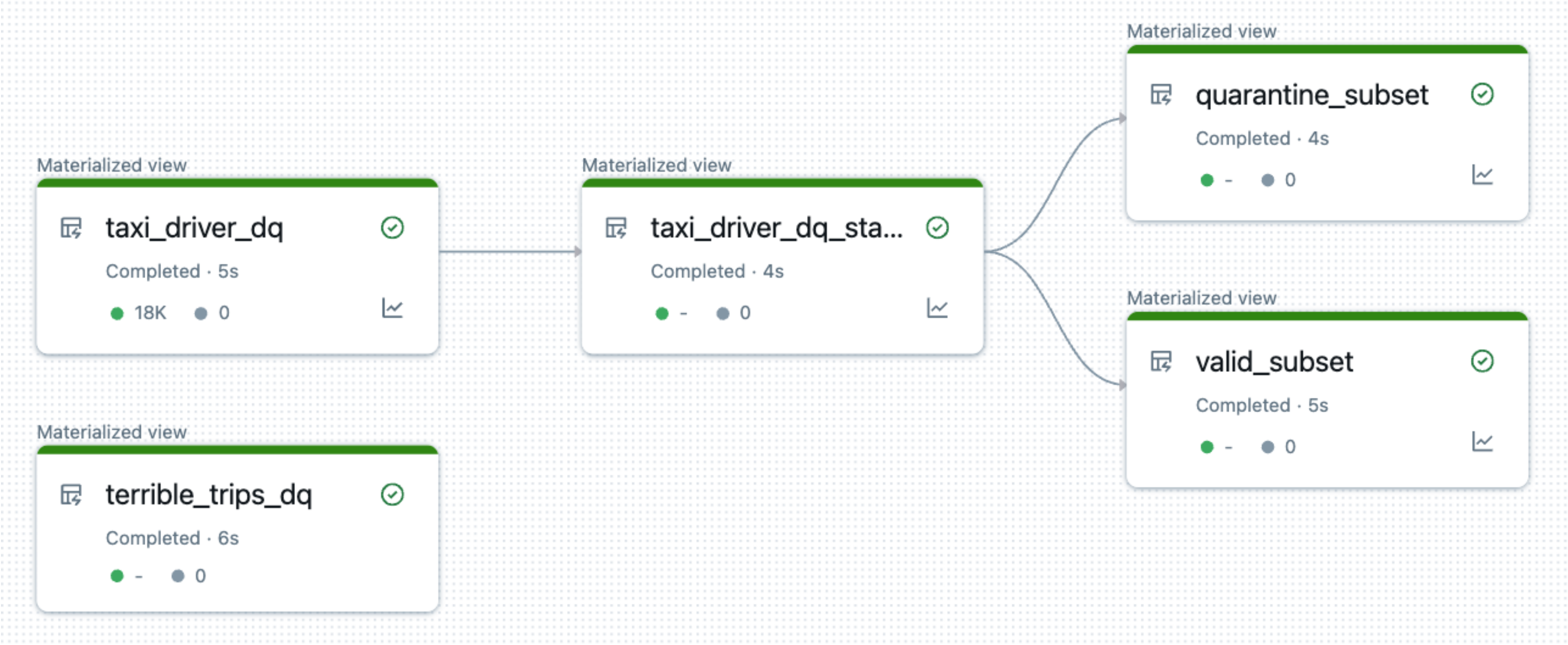
Table

	driver	age	years_driving	state	phone_number
1	Donald Castro	37	9	Colorado	8405400907
2	Brandi Scott	55	32	Iowa	6158471550
3	Kathleen Maldonado	74	24	Washington	269.254.6743
4	Dr. Tyler Watson	53	17	West Virginia	9286680731
5	Brenda Lee	53	21	North Dakota	(681)868-7751
6	Mrs. Melissa Martinez P...	44	22	Hawaii	258.317.2640

How Do You Quarantine A Subset?

```python
@dlt.table()
def quarantine_subset():
    res = dlt.read("taxi_driver_dq_status")
    return res.filter(
        (F.col("age_validity_passed_dq_check") == False)
        | (F.col("state_validity_passed_dq_check") == False)
    )


@dlt.table()
def valid_subset():
    res = dlt.read("taxi_driver_dq_status")
    return res.filter(
        (F.col("age_validity_passed_dq_check") == True)
        & (F.col("state_validity_passed_dq_check") == True)
    )
```

Quarantine Lineage



Quarantine Results

3 minutes ago (16s) 18

```sql
select * from quarantine_subset
```

See performance (1)

Table +

	driver	age	years_driving	state	phone_number	valid_us_state	valid_us_state_abv
1	Jonathan Conley	114	12	New Hampshi...	(994)428-5333	New Hampshire	null
2	Kyle Anderson	105	49	Missouri	596.275.5847x087	Missouri	null
3	Diane White	101	35	Wyoming	547.437.4234	Wyoming	null
4	Jacqueline Morgan	145	19	California	+1-975-434-1649	California	null

Just now (2s) 19

```sql
select * from quarantine_subset
where age_validity_passed_dq_check = True
```

See performance (1)

Table +

	driver	age	years_driving	state	phone_number	valid_us_state	valid_us_state_abv
1	Brian Wong	97	25	null	611-295-2665x21465	null	null
2	Samantha Brown	98	41	oidl	(585)988-4339	null	null
3	Sherri Liu	46	33	null	+1-932-790-8446x32446	null	null
4	Alfred Howell	81	16	null	(946)873-1038	null	null
5	Casey Hunter	35	20	null	(586)622-8460	null	null

Valid Results

4 minutes ago (8s)

```
1  select * from valid_subset
```

See performance (1)

Table

	driver	age	years_driving	state	phone_number
1	Stephanie Hensley	34	30	Montana	(456)626-6413x0113
2	Jonathan Potts	53	27	Missouri	null
3	Alicia Dunn	82	21	North Carolina	833-446-9547x61619
4	Terry Collins	74	21	Alaska	(620)349-4665x960
5	Karen Clay	22	29	Delaware	001-371-874-5539x64647

Extracting Results In Tables

```
CREATE
OR REPLACE TEMP VIEW event_log_raw AS
SELECT
  *
FROM
  event_log("a5ac7ea9-36d6-465a-b3e7-f451c8014011");

CREATE
  OR REPLACE TEMP VIEW latest_update AS
SELECT
  origin.update_id AS id
FROM
  event_log_raw
WHERE
  event_type = 'create_update'
ORDER BY
  timestamp DESC
LIMIT
  1;
```

```
SELECT
  row_expectations.dataset as dataset,
  row_expectations.name as expectation,
  SUM(row_expectations.passed_records) as passing_records,
  SUM(row_expectations.failed_records) as failing_records
FROM
  (
    SELECT
      explode(
        from_json(
          details :flow_progress :data_quality :expectations,
          "array<struct<name: string, dataset: string, passed_records: int, failed_records: int>>"
        )
      ) row_expectations
    FROM
      event_log_raw,
      latest_update
    WHERE
      event_type = 'flow_progress'
      AND origin.update_id = latest_update.id
  )
GROUP BY
  row_expectations.dataset,
  row_expectations.name
```

Expectation Results

	dataset	expectation	passing_records	failing_records
1	taxi_driver_dq	driver_completeness	17713	1
2	terrible_trips_dq	fare_amount_precision	19752	2180
3	terrible_trips_dq	pickup_zip_validity	19775	2157
4	taxi_driver_dq	state_completenss	16824	890
5	taxi_driver_dq	phone_number_validity	3558	14156
6	taxi_driver_dq	driver_uniqueness	17692	22
7	terrible_trips_dq	fare_amount_validity	19102	2830
8	taxi_driver_dq	state_validity	15922	1792
9	terrible_trips_dq	trip_distance_validity	18817	3115
10	terrible_trips_dq	pickup_dropoff_compliance	20812	1120
11	terrible_trips_dq	driver_completeness	20831	1101
12	taxi_driver_dq	age_validity	9573	8141
13	taxi_driver_dq	phone_number_completeness	16855	859
14	taxi_driver_dq	years_driving_validity	14744	2970

Lessons Learned & Subtleties

Which Columns Do You Test Against?

- Testing each column against every dimension can be overkill, especially with large datasets.
- As a developer, choose which columns need data quality expectations and which dimensions to test.
- What if the data changes?
    - Solution: Regular profiling and repeating the cycle.

Overlapping Vs Disjoint Expectations

- Testing a column against multiple dimensions is common.
- Decide in advance how you want to define your expectations:
  - Option 1: A value can fail exactly one dimension
  - Option 2: A value can fail multiple dimensions
- If values can fail multiple dimensions, the number of failures across each dimension can exceed the total due to double counting.

Overlapping Vs Disjoint Expectations II

- It’s not easy to write expectations if you want them to be disjoint.
- Here’s how you might write expectations for the trip distance column to ensure that data can only fail one test.

```
trip_rules["trip_distance_completeness"] = "trip_distance is not null"
trip_rules["trip_distance_validity"] = "trip_distance is null or (trip_distance > 0 and trip_distance < 400)"
trip_rules["trip_distance_precision"] = """
                                        trip_distance is null or
                                        (trip_distance <= 0 or trip_distance <= 400) or
                                        trip_distance = round(trip_distance, 2)
                                        """
```

Strict Vs Loose Expectations

- As the developer, decide how strict or loose to make expectations.
- **State Column Example**:
  - **Strict**: States must be spelled out correctly (e.g., "Massachusetts").
  - **Loose**: Allow variations like "Massachusetts," "MA," "massachusetts."
  - **Two-Word States**:
    - **Strict**: "North Carolina."
    - **Loose**: Variations like "North carolina," "north carolina," "n carolina.”

Strict Vs Loose Expectations II

- Sometimes, very strict rules are not ideal; consider making them looser.
- Example: 'MA' is valid, but a strict check only accepting 'Massachusetts' may result in a low pass rate despite clean data.
- Determining the right strictness level takes practice and iterations.
- Run your pipeline and adjust the strictness as needed.

Make Use Of SQL Functions

- The `fare_amount` column has values with more than 2 decimal places.
- This doesn't make sense, so how do you flag these values?
- **Solution**: Use the ROUND function.
  - Example: `fare_amount = ROUND(fare_amount, 2)`.

	tpep_pickup_datetime	tpep_dropoff_datetime	trip_distance	fare_amount	pickup_zip	dropoff_zip	driver
1	2016-02-20T22:09:04.000+00:00	2016-02-20T22:22:43.000+00:00	9	35.44	64714	10003	null
2	2016-02-17T15:54:42.000+00:00	2016-02-17T16:04:41.000+00:00	17	50.3	57000	10021	null
3	2016-02-19T07:47:10.000+00:00	2016-02-19T07:57:32.000+00:00	13	36.74892	47424	10171	null
4	2016-02-23T06:56:26.000+00:00	2016-02-23T07:00:30.000+00:00	7	54.2	57759	10199	null

Try Regular Expressions

- Phone number columns are difficult to test for validity.
- Use a regular expression to help with this.
- Adjust the regular expression based on the strictness of your validity test.
  - Example: For (xxx) xxx-xxxx, use a simple regular expression.
  - Allowing country codes or plain 10-digit strings makes the expression more complex.

Summary and Key Takeaways

- **Importance of Data Quality**: Ensuring data quality is crucial for accurate analytics, decision-making, and building reliable AI/ML models.
- **Defining Data Expectations**: Data expectations are rules that specify how data should look, similar to unit tests for data.
- **Steps to Write Expectations**: Profile your data, consult with domain experts, document rules, and transform them into SQL statements.
- **Tips**: Iterate over the data expectation steps to ensure the best outcomes.

Q&A

References I

Gartner. "12 Actions to Improve Your Data Quality." Gartner, 14 July 2021. https://www.gartner.com/smarterwithgartner/how-to-improve-your-data-quality.

Databricks. "Data Quality: Ensure Accuracy for Better Decisions." Databricks, 2023. https://www.databricks.com/glossary/data-quality.

Terry, Matt. "Unlocking Enterprise AI." *Economist Impact*, November 22, 2024. https://impact.economist.com/perspectives/technology-innovation/unlocking-enterprise-ai.

IBM. "What Is Data Quality?" *IBM*, 2021. https://www.ibm.com/topics/data-quality.

Gschwind, Mark. "Enterprise Information Management (EIM) in SQL Server 2012." SlideShare. February 25, 2013. https://www.slideshare.net/slideshow/gschwind-mdsdqs-sqlsatmv/16761590#4.

Databricks. "Manage Data Quality with Pipeline Expectations." Databricks Documentation. Last modified January 17, 2025. Accessed January 22, 2025. https://docs.databricks.com/en/delta-live-tables/expectations.html#manage-data-quality-with-pipeline-expectations.

References II

Extreme Weather Watch. "Lowest Temperatures in San Francisco History." Accessed January 28, 2025. https://www.extremeweatherwatch.com/cities/san-francisco/lowest-temperatures.

NYC Taxi & Limousine Commission. "Taxi Fare Information." NYC.gov. Accessed January 28, 2025. https://www.nyc.gov/site/tlc/passengers/taxi-fare.page.

NYC Taxi & Limousine Commission. "Get a TLC Driver's License." NYC.gov. Accessed January 28, 2025. https://www.nyc.gov/site/tlc/drivers/get-a-tlc-drivers-license.page.

NYC Taxi & Limousine Commission. "Taxi Fare Information." NYC.gov. Accessed January 28, 2025. https://www.nyc.gov/site/tlc/passengers/taxi-fare.page.

References III

Meier, Eric. "What is the Longest Road Trip Possible in New York State?" Lite 98.7. Accessed January 28, 2025. https://lite987.com/longest-roadtrip-possible-in-new-york/.

Databricks. "Quarantine Invalid Records." Databricks Documentation. Last modified January 17, 2025. Accessed January 22, 2025. https://docs.databricks.com/en/delta-live-tables/expectation-patterns.html#quarantine-invalid-records.

Databricks. "Quarantine Invalid Records." Databricks Documentation. Last modified January 17, 2025. Accessed January 22, 2025. https://docs.databricks.com/en/delta-live-tables/expectation-patterns.html#quarantine-invalid-records.