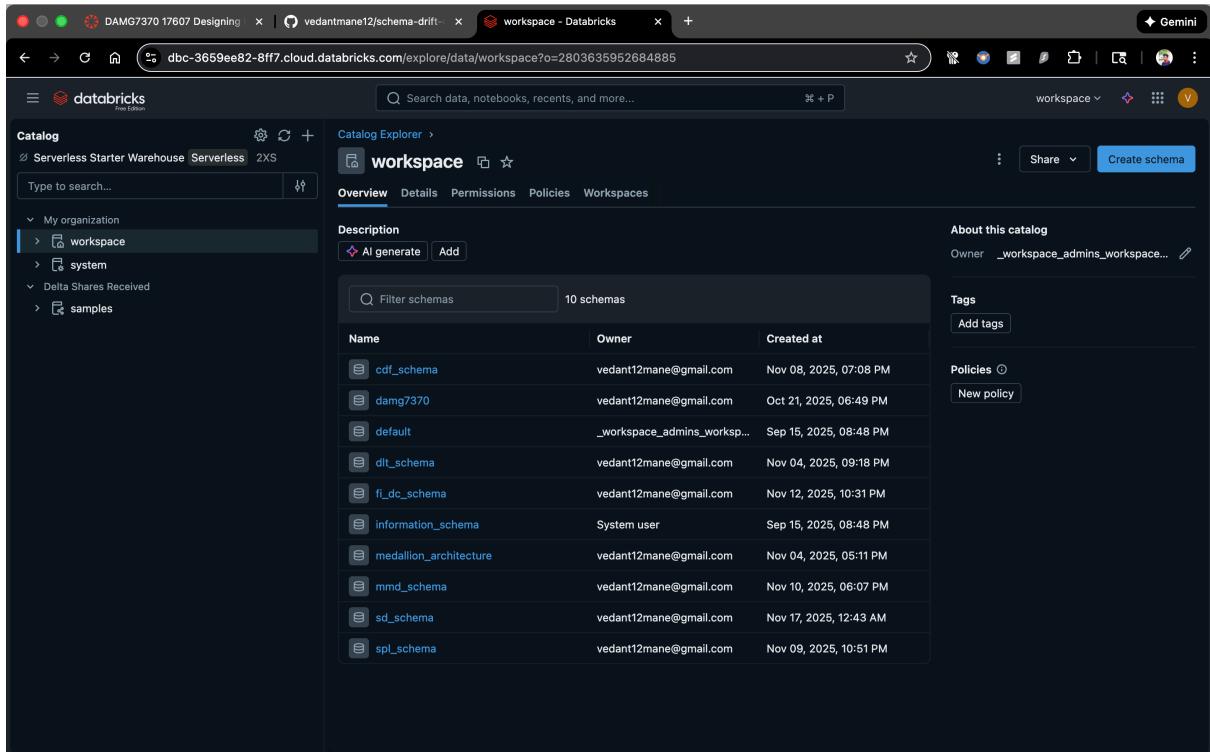


Assignment 09 - Schema Drift Implementation using Python in Databricks

Catalog: workspace



The screenshot shows the Databricks Catalog Explorer interface. The left sidebar shows 'My organization' with 'workspace' selected. The main area displays a table of schemas under the 'workspace' catalog. The table has columns for Name, Owner, and Created at. The schemas listed are:

Name	Owner	Created at
cdf_schema	vedant12mane@gmail.com	Nov 08, 2025, 07:08 PM
damg7370	vedant12mane@gmail.com	Oct 21, 2025, 06:49 PM
default	_workspace_admins_workspace...	Sep 15, 2025, 08:48 PM
dit_schema	vedant12mane@gmail.com	Nov 04, 2025, 09:18 PM
fi_dc_schema	vedant12mane@gmail.com	Nov 12, 2025, 10:31 PM
information_schema	System user	Sep 15, 2025, 08:48 PM
medallion_architecture	vedant12mane@gmail.com	Nov 04, 2025, 05:11 PM
mmd_schema	vedant12mane@gmail.com	Nov 10, 2025, 06:07 PM
sd_schema	vedant12mane@gmail.com	Nov 17, 2025, 12:43 AM
spl_schema	vedant12mane@gmail.com	Nov 09, 2025, 10:51 PM

Schema: workspace.sd_schema

Catalog Explorer > workspace > sd_schema

Overview Details Permissions Policies

Description

Name Owner Created at

datastore	vedant12mane@gmail.com	Nov 17, 2025, 12:44 AM
-----------	------------------------	------------------------

About this schema

Owner vedant12mane@gmail.com

Tags

Policies

Pipeline Run: customer_data_1.json

Volume: workspace.sd_schema.datastore

Catalog Explorer > workspace > sd_schema > datastore

Overview Files Details Permissions

Description

/Volumes/workspace/sd_schema/datastore

Name Size Last modified

customer_data_1.json	1.03 KB	10 hours ago
----------------------	---------	--------------

About this volume

Owner vedant12mane@gmail.com

Tags

Bronze Layer Implementation (Rescue Data)

Screenshot of Databricks workspace showing the implementation of the Silver layer (Rescue Data). The notebook `SchemaDrift_PL` contains Python code for reading raw data from a cloud file and writing it to a bronze table. The pipeline graph shows a flow from a streaming table `cust_bronze_sd_rescue` to another table `cust_silver_sd_rescue`. The table view shows 5 rows of sample data.

```

4 # Ingest the raw data into the bronze table using append flow
5 @pl.append_flow(
6     target = "cust_bronze_sd_rescue", #object name
7     name = "cust_bronze_sd_rescue_ingest_flow" #flow name
8 )
9 def cust_bronze_sd_rescue_ingest_flow():
10     df = (
11         spark.readStream
12             .format("cloudFiles")
13             .option("cloudFiles.format", "json")
14             .option("cloudFiles.inferColumnTypes", "true") #auto scan schema
15             .option("cloudFiles.schemaEvolutionMode", "failOnNewColumns") # schema customer_data_1.json is
16             #different than customer_data_2.json so it fails with [UNKNOWN_FIELD_EXCEPTION].
17             .option("cloudFiles.schemaEvolutionMode", "rescue")
18             .load(f"volume path")
19     )

```

	A _c City	A _c CustomerID	A _c Email	A _c FullName	A _c PhoneNumber	A _c SignupDate	A _c _rescued_data	E _c Ingestion_datetime	A _c source_filename
1	New York	C001	alice.j@example.com	Alice Johnson	555-123-4567	2023-01-15	null	2025-11-17T20:55:38.773+00:00	> /Volumes/workspace/
2	Chicago	C002	bob.smith@example.co...	Bob Smith	555-234-5678	2023-02-20	null	2025-11-17T20:55:38.773+00:00	> /Volumes/workspace/
3	San Diego	C003	carol.lee@example.com	Carol Lee	555-345-6789	2023-03-05	null	2025-11-17T20:55:38.773+00:00	> /Volumes/workspace/
4	Austin	C004	david.kim@example.com	David Kim	555-456-7890	2023-04-12	null	2025-11-17T20:55:38.773+00:00	> /Volumes/workspace/
5	Dallas	C010	jack.n@example.com	Jack Nguyen	555-012-3456	2023-10-21	null	2025-11-17T20:55:38.773+00:00	> /Volumes/workspace/

Silver Layer Implementation (Rescue Data)

Screenshot of Databricks workspace showing the implementation of the Silver layer (Rescue Data). The notebook `SchemaDrift_PL` contains Python code for creating a streaming table and an append flow. The pipeline graph shows a flow from a table `cust_silver_sd_rescue` to another table `cust_silver_sd_rescue`. The table view shows 5 rows of sample data.

```

# plain implementation without processing _rescued_data field. Use this when you upload customer_data_1.json
#
pl.create_streaming_table(
    name = "cust_silver_sd_rescue",
    expect_all_or_drop = ("no_rescued_data": "_rescued_data IS NULL", "valid_id": "CustomerID IS NOT NULL")
)
@pl.append_flow(
    target = "cust_silver_sd_rescue",
    name = "cust_silver_sd_rescue_clean_flow"
)
def cust_silver_sd_rescue_clean_flow():
    return (
        spark.readStream.table("cust_bronze_sd_rescue")
    )

```

	A _c City	A _c CustomerID	A _c Email	A _c FullName	A _c PhoneNumber	A _c SignupDate	A _c _rescued_data	E _c Ingestion_datetime	A _c source_filename
1	New York	C001	alice.j@example.com	Alice Johnson	555-123-4567	2023-01-15	null	2025-11-17T20:55:38.773+00:00	> /Volumes/workspace/
2	Chicago	C002	bob.smith@example.co...	Bob Smith	555-234-5678	2023-02-20	null	2025-11-17T20:55:38.773+00:00	> /Volumes/workspace/
3	San Diego	C003	carol.lee@example.com	Carol Lee	555-345-6789	2023-03-05	null	2025-11-17T20:55:38.773+00:00	> /Volumes/workspace/
4	Austin	C004	david.kim@example.com	David Kim	555-456-7890	2023-04-12	null	2025-11-17T20:55:38.773+00:00	> /Volumes/workspace/
5	Dallas	C010	jack.n@example.com	Jack Nguyen	555-012-3456	2023-10-21	null	2025-11-17T20:55:38.773+00:00	> /Volumes/workspace/

Bronze Layer Implementation (AddNewColumns Data)

`#bronze layer table: cust_bronze_sd
pl.create_streaming_table("cust_bronze_sd_addNew")`

`# Ingest the raw data into the bronze table using append flow
@pl.append_flow(
 target = "cust_bronze_sd_addNew", #object name
 name = "cust_bronze_sd_addNew_ingest_flow" #flow name
)
def cust_bronze_sd_addNew_ingest_flow():
 #...`

	City	CustomerID	Email	FullName	PhoneNumber	SignupDate	_rescued_data	Ingestion_datetime	source_filename
1	New York	C001	alice.j@example.com	Alice Johnson	555-123-4567	2023-01-15	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/
2	Chicago	C002	bob.smith@example.co...	Bob Smith	555-234-5678	2023-02-20	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/
3	San Diego	C003	carol.lee@example.com	Carol Lee	555-345-6789	2023-03-05	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/
4	Austin	C004	david.kim@example.com	David Kim	555-456-7890	2023-04-12	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/
5	Dallas	C010	jack.n@example.com	Jack Nguyen	555-012-3456	2023-10-21	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/

Silver Layer Implementation (AddNewColumns Data)

`#bronze layer table: cust_bronze_sd
pl.create_streaming_table("cust_bronze_sd_addNew")`

`# Ingest the raw data into the bronze table using append flow
@pl.append_flow(
 target = "cust_silver_sd_address", #object name
 name = "New_ingest_flow" #flow name
)
def cust_bronze_sd_new_ingest_flow():
 #...`

	City	CustomerID	Email	FullName	PhoneNumber	SignupDate	_rescued_data	Ingestion_datetime	source_filename
1	New York	C001	alice.j@example.com	Alice Johnson	555-123-4567	2023-01-15	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/
2	Chicago	C002	bob.smith@example.co...	Bob Smith	555-234-5678	2023-02-20	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/
3	San Diego	C003	carol.lee@example.com	Carol Lee	555-345-6789	2023-03-05	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/
4	Austin	C004	david.kim@example.com	David Kim	555-456-7890	2023-04-12	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/
5	Dallas	C010	jack.n@example.com	Jack Nguyen	555-012-3456	2023-10-21	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/

Pipeline Run: customer_data_2.json

Volume: workspace.sd_schema.datastore

The screenshot shows the Databricks Catalog Explorer interface. On the left, there's a sidebar with a tree view of the catalog. The 'datastore' volume is selected. The main area shows the contents of the 'datastore' volume, which includes two JSON files: 'customer_data_1.json' and 'customer_data_2.json'. There are buttons for 'Create directory' and 'Add tags'.

Pipeline Run: (Normal)

The screenshot shows the Databricks Notebook editor. A pipeline named 'SchemaDrift_PL' is being run. The notebook code includes logic for schema evolution. The right panel displays the 'Pipeline graph' section, which is currently unavailable. An error message states: 'Pipeline initialization failed' and 'An error occurred during pipeline initialization'. Below the graph, there's a 'View details' button and a 'View logs' button.

Pipeline Run: (Full Table Refresh)

The screenshot shows a Databricks notebook titled "SchemaDrift_PL" with a "schema-drift-databricks-implementation" workspace. A modal window titled "Full refresh" is open, explaining that it will truncate and recompute ALL tables in the pipeline from scratch. It asks if the user is sure they want to proceed. Below the modal, a note discusses schema evolution. The notebook code includes options for "cloudFiles.schemaEvolutionMode" and "failOnNewColumns". The notebook interface shows tables and logs. A prominent error message in the top right corner states "Pipeline initialization failed" with the sub-message "An error occurred during pipeline initialization".

The screenshot shows the "Jobs & Pipelines" section of the Databricks interface. A specific pipeline named "SchemaDrift_PL" is selected. The pipeline details pane on the right shows the following information:

- Pipeline ID:** 77f12922-8c4c-44a6-939f-9f19e7667cef
- Pipeline type:** ETL pipeline
- Source code:** /Workspace/Users/vedant12mane@gmail.com/schema-drift-databricks/SchemaDrift_PL
- Run as:** vedant12mane@gmail.com
- Tags:** None

The main area displays a graph of two streaming tables connected by a flow. The top table is "cust_bronze_sd_r..." and the bottom table is "cust_silver_sd_res...". Both tables show completion metrics: 4s for cust_bronze and 3s for cust_silver. An event log at the bottom shows recent pipeline activity.

Bronze Layer Implementation (Rescue Data)

```

#bronze layer table: cust_bronze_sd
pl.create_streaming_table("cust_bronze_sd_rescue")

# Ingest the raw data into the bronze table using append flow
@pl.append_flow
@target = "cust_bronze_sd_rescue", #object name
name = "cust_bronze_sd_rescue ingest flow" #flow name

```

Last update: Nov 17, 2025, 4:52:54 PM

	Age	City	CustomerID	Email	FullName	Gender	LoyaltyStatus	PhoneNumber	SignupDate	_rescued_data
1	null	New York	C001	alice.j@example.com	Alice Johnson	null	null	555-123-4567	2023-01-15	null
2	null	Chicago	C002	bob.smith@example.com	Bob Smith	null	null	555-234-5678	2023-02-20	null
3	null	San Diego	C003	carol.lee@example.com	Carol Lee	null	null	555-345-6789	2023-03-05	null
4	null	Austin	C004	david.kim@example.com	David Kim	null	null	555-456-7890	2023-04-12	null
5	null	Dallas	C010	jack.n@example.com	Jack Nguyen	null	null	555-012-3456	2023-10-21	null
6	26	New York	C001	alice.johnson@example.co...	Alice Johnson	Female	Platinum	555-116-7521	2023-02-28	null
7	58	Chicago	C002	bob.smith@example.com	Bob Smith	Male	Silver	555-534-5537	2023-08-04	null
8	34	San Diego	C003	carol.lee@example.com	Carol Lee	Female	Platinum	555-524-5491	2023-05-24	null
9	66	Austin	C004	david.kim@example.com	David Kim	Non-binary	Bronze	555-557-5139	2023-03-11	null
10	34	Seattle	C005	eva.martinez@example.com	Eva Martinez	Female	Platinum	555-384-8895	2023-04-05	null
11	34	Denver	C006	frank.wright@example.com	Frank Wright	Non-binary	Bronze	555-392-5331	2023-09-29	null
12	38	Boston	C007	grace.chen@example.com	Grace Chen	Female	Bronze	555-570-7081	2023-09-18	null
13	29	Miami	C008	henry.patel@example.com	Henry Patel	Non-binary	Platinum	555-115-6962	2023-06-17	null

Silver Layer Implementation (Rescue Data)

```

# Function to handle adding NEW FIELDS
def process_rescue_data_new_fields(df):
    #Add all fields from _rescued_data to key map
    df = df.withColumn(
        "_rescued_data_json_to_map",
        from_json(
            col("_rescued_data"),
            schema=StructType([
                ...
            ])
        ).alias("_rescued_data_json_to_map")
    )

```

Last update: Nov 17, 2025, 4:52:54 PM

	Age	City	CustomerID	Email	FullName	Gender	LoyaltyStatus	PhoneNumber	signupDate	_rescued_data
1	null	New York	C001	alice.j@example.com	Alice Johnson	null	null	555-123-4567	2023-01-15	null
2	null	Chicago	C002	bob.smith@example.com	Bob Smith	null	null	555-234-5678	2023-02-20	null
3	null	San Diego	C003	carol.lee@example.com	Carol Lee	null	null	555-345-6789	2023-03-05	null
4	null	Austin	C004	david.kim@example.com	David Kim	null	null	555-456-7890	2023-04-12	null
5	null	Dallas	C010	jack.n@example.com	Jack Nguyen	null	null	555-012-3456	2023-10-21	null
6	26	New York	C001	alice.johnson@example.co...	Alice Johnson	Female	Platinum	555-116-7521	2023-02-28	null
7	58	Chicago	C002	bob.smith@example.com	Bob Smith	Male	Silver	555-534-5537	2023-08-04	null
8	34	San Diego	C003	carol.lee@example.com	Carol Lee	Female	Platinum	555-524-5491	2023-05-24	null
9	66	Austin	C004	david.kim@example.com	David Kim	Non-binary	Bronze	555-557-5139	2023-03-11	null
10	34	Seattle	C005	eva.martinez@example.com	Eva Martinez	Female	Platinum	555-384-8895	2023-04-05	null
11	34	Denver	C006	frank.wright@example.com	Frank Wright	Non-binary	Bronze	555-392-5331	2023-09-29	null
12	38	Boston	C007	grace.chen@example.com	Grace Chen	Female	Bronze	555-570-7081	2023-09-18	null
13	29	Miami	C008	henry.patel@example.com	Henry Patel	Non-binary	Platinum	555-115-6962	2023-06-17	null

Bronze Layer Implementation (AddNewColumns Data)

```

    col("rescued_data"),
    MapType(StringType(), StringType())
)
Catalog: workspace
Schema: sd_schema
# Extract all keys from rescued_data map keys
    
```

Last update: Nov 17, 2025, 11:47:13 PM

	PhoneNumber	SignupDate	_rescued_data	ingestion_datetime	source_filename	Age	Gender	LoyaltyStatus
1	555-116-7521	2023-02-28	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	26	Female	Platinum
2	555-534-5537	2023-08-04	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	58	Male	Silver
3	555-524-5491	2023-05-24	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	34	Female	Platinum
4	555-557-5139	2023-03-11	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	66	Non-binary	Bronze
5	555-384-8895	2023-04-05	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	34	Female	Platinum
6	555-392-5331	2023-09-29	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	34	Non-binary	Bronze
7	555-570-7081	2023-09-18	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	38	Female	Bronze
8	555-115-6962	2023-06-17	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	29	Non-binary	Platinum
9	555-123-4567	2023-01-15	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/sd_schema/datas...	null	null	null
10	555-234-5678	2023-02-20	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/sd_schema/datas...	null	null	null
11	555-345-6789	2023-03-05	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/sd_schema/datas...	null	null	null
12	555-456-7890	2023-04-12	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/sd_schema/datas...	null	null	null
13	555-012-3456	2023-10-21	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/sd_schema/datas...	null	null	null

Silver Layer Implementation (AddNewColumns Data)

```

    col("rescued_data"),
    MapType(StringType(), StringType())
)
Catalog: workspace
Schema: sd_schema
# Extract all keys from rescued_data map keys
    
```

Last update: Nov 17, 2025, 11:47:13 PM

	FullName	PhoneNumber	SignupDate	_rescued_data	ingestion_datetime	source_filename	Age	Gender
1	Alice Johnson	555-116-7521	2023-02-28	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	26	Female
2	Bob Smith	555-534-5537	2023-08-04	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	58	Male
3	Carol Lee	555-524-5491	2023-05-24	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	34	Female
4	David Kim	555-557-5139	2023-03-11	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	66	Non-binary
5	Eva Martinez	555-384-8895	2023-04-05	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	34	Female
6	Frank Wright	555-392-5331	2023-09-29	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	34	Non-binary
7	Grace Chen	555-570-7081	2023-09-18	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	38	Female
8	Henry Patel	555-115-6962	2023-06-17	null	2025-11-18T04:46:34.847+00:00	/Volumes/workspace/sd_schema/datas...	29	Non-binary
9	Alice Johnson	555-123-4567	2023-01-15	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/sd_schema/datas...	null	null
10	Bob Smith	555-234-5678	2023-02-20	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/sd_schema/datas...	null	null
11	Carol Lee	555-345-6789	2023-03-05	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/sd_schema/datas...	null	null
12	David Kim	555-456-7890	2023-04-12	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/sd_schema/datas...	null	null
13	Jack Nguyen	555-012-3456	2023-10-21	null	2025-11-18T04:44:24.794+00:00	/Volumes/workspace/sd_schema/datas...	null	null

Pipeline Run: customer_data_3.json

Volume: workspace.sd_schema.datasource

The screenshot shows the Databricks workspace with the SchemaDrift_PL notebook open. The Pipeline graph on the right displays a single step named "cust_bronze_sd_rescue" which outputs 9 records to a "Streaming table" named "cust_silver_sd_rescue". The "cust_silver_sd_rescue" table has 9 output records and 2 expectations. Below the graph, a table view shows 22 rows of data with columns: Age, City, CustomerID, Email, FullName, Gender, LoyaltyStatus, PhoneNumber, SignupDate, and _rescued_. The data is identical to the Bronze layer implementation.

Bronze Layer Implementation (Rescue Data)

The screenshot shows the Databricks workspace with the SchemaDrift_PL notebook open. The Pipeline graph on the right displays a single step named "cust_bronze_sd_rescue" which outputs 9 records to a "Streaming table" named "cust_silver_sd_rescue". The "cust_silver_sd_rescue" table has 9 output records and 2 expectations. Below the graph, a table view shows 22 rows of data with columns: Age, City, CustomerID, Email, FullName, Gender, LoyaltyStatus, PhoneNumber, SignupDate, and _rescued_. The data is identical to the Bronze layer implementation.

Silver Layer Implementation (Rescue Data)

```

1 #bronze layer table: cust_bronze_sd
2 pl.create_streaming_table("cust_bronze_sd_rescue")
3
4 # Ingest the raw data into the bronze table using append flow
5 @pl.append_flow(
6   +format="json",
7   +source="Mouse browser or救援表"

```

	Age	City	CustomerID	Email	FullName	Gender	LoyaltyStatus	PhoneNumber	signupDate	_rescued_
9	66	Austin	C004	david.kim@example.com	David Kim	Non-binary	Bronze	555-557-5139	2023-03-11	null
10	34	Seattle	C005	eva.martinez@example.com	Eva Martinez	Female	Platinum	555-384-8895	2023-04-05	null
11	34	Denver	C006	frank.wright@example.com	Frank Wright	Non-binary	Bronze	555-392-5331	2023-09-29	null
12	38	Boston	C007	grace.chen@example.com	Grace Chen	Female	Bronze	555-570-7081	2023-09-18	null
13	29	Miami	C008	henry.patel@example.com	Henry Patel	Non-binary	Platinum	555-115-6962	2023-06-17	null
14	49	North William	C001	huntsamantha@example.com	Michael Webb	Male	Platinum	001-711-328-0096	2024-04-10	null
15	60	East Christopherview	C002	susanwilson@example.com	Chris Hensley	Female	Gold	001-787-381-7723	2024-09-11	null
16	22	Jamesstad	C003	taylorbarr@example.net	Courtney White	Male	Bronze	3402852594	2024-02-18	null
17	52	Port Joanna	C004	thayes@example.net	Cynthia Mills	Male	Gold	694-884-5528x7633	2024-08-26	null
18	61	Grantborough	C005	chadanderson@example.com	Sandra Taylor	Male	Platinum	5194474151	2024-07-18	null
19	21	New Garrett	C006	joshua55@example.net	Kimberly Daugherty	Female	Platinum	679-741-5908x091	2025-01-12	null
20	44	Lake Bryan	C007	lglover@example.org	Lindsey McGuire	Male	Platinum	342-569-7735x921	2024-07-23	null
21	29	Taylorview	C009	ucoffee@example.net	Caroline Morris	Male	Gold	+1-738-592-5919x344	2024-09-28	null
22	55	Woodsport	C011	matthewthomas@example.net	Benjamin Fernand...	Male	Bronze	520-274-1325	2024-09-18	null

Bronze Layer Implementation (AddNewColumns Data)

```

map_keys(col("_rescued_data_json_to_map"))
  .alias("rescued_key")
  .distinct()

# Collect keys as a list (only if df is not streaming)
# If streaming, you must provide the list of possible keys another way
new_keys = [row["rescued_key"] for row in df.keys.collect()] if not df.isStreaming else []

```

	City	CustomerID	Email	FullName	PhoneNumber	SignupDate	_rescued_data	ingestion_datetime	source
1	North William	C001	huntsamantha@example.com	Michael Webb	001-711-328-0096	2024-04-10	null	2025-11-18T04:51:04.380+00:00	/Volume
2	East Christopherview	C002	susanwilson@example.org	Chris Hensley	001-787-381-7723	2024-09-11	null	2025-11-18T04:51:04.380+00:00	/Volume
3	Jamesstad	C003	taylorbarr@example.net	Courtney White	3402852594	2024-02-18	null	2025-11-18T04:51:04.380+00:00	/Volume
4	Port Joanna	C004	thayes@example.net	Cynthia Mills	694-884-5528x7633	2024-08-26	null	2025-11-18T04:51:04.380+00:00	/Volume
5	Grantborough	C005	chadanderson@example.com	Sandra Taylor	5194474151	2024-07-18	null	2025-11-18T04:51:04.380+00:00	/Volume
6	New Garrett	C006	joshua55@example.net	Kimberly Daugherty	679-741-5908x091	2025-01-12	null	2025-11-18T04:51:04.380+00:00	/Volume
7	Lake Bryan	C007	lglover@example.org	Lindsey McGuire	342-569-7735x921	2024-07-23	null	2025-11-18T04:51:04.380+00:00	/Volume
8	Taylorview	C009	ucoffee@example.net	Caroline Morris	+1-738-592-5919x344	2024-09-28	null	2025-11-18T04:51:04.380+00:00	/Volume
9	Woodsport	C011	matthewthomas@example.net	Benjamin Fernand...	520-274-1325	2024-09-18	null	2025-11-18T04:51:04.380+00:00	/Volume
10	New York	C001	alice.johnson@example.com	Alice Johnson	555-116-7521	2023-02-28	null	2025-11-18T04:46:34.847+00:00	/Volume
11	Chicago	C002	bob.smith@example.com	Bob Smith	555-534-5537	2023-08-04	null	2025-11-18T04:46:34.847+00:00	/Volume
12	San Diego	C003	carol.lee@example.com	Carol Lee	555-524-5491	2023-05-24	null	2025-11-18T04:46:34.847+00:00	/Volume
13	Austin	C004	david.kim@example.com	David Kim	555-557-5139	2023-03-11	null	2025-11-18T04:46:34.847+00:00	/Volume

Silver Layer Implementation (AddNewColumns Data)

The screenshot shows a Databricks notebook titled "SchemaDrift_PL" with a Python script for "addNewColumns". The code maps keys from rescued data and distinct rows. The Pipeline graph shows a single step named "cust_silver_sd_address" with 9 output records. Below the notebook is a table view of the data.

	City	CustomerID	Email	FullName	PhoneNumber	SignupDate	_rescued_data	ingestion_datetime	source
1	North William	C001	huntsamatha@example.com	Michael Webb	001-711-328-0096	2024-04-10	null	2025-11-18T04:51:04.380+00:00	/Volume
2	East Christopherview	C002	susanwilson@example.org	Chris Hensley	001-787-381-7723	2024-09-11	null	2025-11-18T04:51:04.380+00:00	/Volume
3	Jamesstad	C003	taylorbar@example.net	Courtney White	3402852594	2024-02-18	null	2025-11-18T04:51:04.380+00:00	/Volume
4	Port Joanna	C004	thayes@example.net	Cynthia Mills	694-884-5528x7633	2024-08-26	null	2025-11-18T04:51:04.380+00:00	/Volume
5	Grantborough	C005	chadanderson@example.com	Sandra Taylor	5194474151	2024-07-18	null	2025-11-18T04:51:04.380+00:00	/Volume
6	New Garrett	C006	joshua55@example.net	Kimberly Daugherty	679-741-5908x091	2025-01-12	null	2025-11-18T04:51:04.380+00:00	/Volume
7	Lake Bryan	C007	lglover@example.org	Lindsey McGuire	342-569-7735x921	2024-07-23	null	2025-11-18T04:51:04.380+00:00	/Volume
8	Taylorview	C009	ucoffee@example.net	Caroline Morris	+1-738-592-6919x344	2024-09-28	null	2025-11-18T04:51:04.380+00:00	/Volume
9	Woodsport	C011	matthewthomas@example.net	Benjamin Fernand...	520-274-1325	2024-09-18	null	2025-11-18T04:51:04.380+00:00	/Volume
10	New York	C001	alice.johnson@example.com	Alice Johnson	555-116-7521	2023-02-28	null	2025-11-18T04:46:34.847+00:00	/Volume
11	Chicago	C002	bob.smith@example.com	Bob Smith	555-534-5537	2023-08-04	null	2025-11-18T04:46:34.847+00:00	/Volume
12	San Diego	C003	carol.lee@example.com	Carol Lee	555-524-5491	2023-05-24	null	2025-11-18T04:46:34.847+00:00	/Volume
13	Austin	C004	david.kim@example.com	David Kim	555-557-5139	2023-03-11	null	2025-11-18T04:46:34.847+00:00	/Volume
...
22 rows	Query performance								

Pipeline Run: customer_data_4.json

Volume: workspace.sd_schema.datastore

The screenshot shows the Databricks Catalog Explorer for the workspace.sd_schema.datastore volume. It contains four files: customer_data_1.json, customer_data_2.json, customer_data_3.json, and customer_data_4.json.

Name	Size	Last modified
customer_data_1.json	1.03 KB	16 hours ago
customer_data_2.json	2.32 KB	20 minutes ago
customer_data_3.json	2.09 KB	9 minutes ago
customer_data_4.json	2.17 KB	23 seconds ago

Bronze Layer Implementation (Rescue Data)

Screenshot of Databricks Notebook showing the implementation of the Bronze Layer (SchemaDrift_PL notebook). The code reads from a Kafka stream named 'cust_bronze_sd_rescue'.

```

    cust_bronze_sd_rescue
    Catalog: workspace
    Schema: sd_schema
    k.readStream.table("cust_bronze_sd_rescue")

```

The resulting table 'cust_bronze_sd_rescue' contains the following data:

Status	PhoneNumber	SignupDate	_rescued_data	ingestion_datetime	source_filename	CreditScore
17	694-884-5528x7633	2024-08-26	null	2025-11-18T04:59:03.542+00:00	/Volumes/workspace/sd_schema/datas...	null
18	5194474151	2024-07-18	null	2025-11-18T04:59:03.542+00:00	/Volumes/workspace/sd_schema/datas...	null
19	679-741-5908x091	2025-01-12	null	2025-11-18T04:59:03.542+00:00	/Volumes/workspace/sd_schema/datas...	null
20	342-569-7735x921	2024-07-23	null	2025-11-18T04:59:03.542+00:00	/Volumes/workspace/sd_schema/datas...	null
21	+1-738-592-5919x344	2024-09-28	null	2025-11-18T04:59:03.542+00:00	/Volumes/workspace/sd_schema/datas...	null
22	520-274-1325	2024-09-18	null	2025-11-18T04:59:03.542+00:00	/Volumes/workspace/sd_schema/datas...	null
23	555-980-4337	null	{"CreditScore": 822, "file_path": "/Volumes/w...	2025-11-18T05:00:12.363+00:00	/Volumes/workspace/sd_schema/datas...	null
24	555-916-4679	null	{"CreditScore": 711, "file_path": "/Volumes/w...	2025-11-18T05:00:12.363+00:00	/Volumes/workspace/sd_schema/datas...	null
25	555-621-5430	null	{"CreditScore": 610, "file_path": "/Volumes/w...	2025-11-18T05:00:12.363+00:00	/Volumes/workspace/sd_schema/datas...	null
26	555-959-9638	null	{"CreditScore": 589, "file_path": "/Volumes/w...	2025-11-18T05:00:12.363+00:00	/Volumes/workspace/sd_schema/datas...	null
27	555-116-5138	null	{"CreditScore": 552, "file_path": "/Volumes/w...	2025-11-18T05:00:12.363+00:00	/Volumes/workspace/sd_schema/datas...	null
28	555-999-9453	null	{"CreditScore": 510, "file_path": "/Volumes/w...	2025-11-18T05:00:12.363+00:00	/Volumes/workspace/sd_schema/datas...	null
29	555-416-7540	null	{"CreditScore": 712, "file_path": "/Volumes/w...	2025-11-18T05:00:12.363+00:00	/Volumes/workspace/sd_schema/datas...	null
30	555-640-2842	null	{"CreditScore": 801, "file_path": "/Volumes/w...	2025-11-18T05:00:12.363+00:00	/Volumes/workspace/sd_schema/datas...	null
31	555-795-7023	null	{"CreditScore": 520, "file_path": "/Volumes/w...	2025-11-18T05:00:12.363+00:00	/Volumes/workspace/sd_schema/datas...	null
32	555-298-1940	null	{"CreditScore": 482, "file_path": "/Volumes/w...	2025-11-18T05:00:12.363+00:00	/Volumes/workspace/sd_schema/datas...	null

Silver Layer Implementation (Rescue Data)

Screenshot of Databricks Notebook showing the implementation of the Silver Layer (SchemaDrift_PL notebook). The code defines a schema for the 'cust_silver_sd_rescue' table.

```

    StructField("signupDate", DateType(), True),
    StructField("age", IntegerType(), True),
    StructField("CreditScore", IntegerType(), True)
  }

  expected_new_fields = [
    # "age",
    # "gender",
    # "loyaltyStatus",
  ]

```

The resulting table 'cust_silver_sd_rescue' contains the following data:

Gender	LoyaltyStatus	PhoneNumber	SignupDate	_rescued_data	ingestion_datetime	source_filename	CreditScore
Female	Bronze	520-274-1325	2024-09-18	null	2025-11-18T05:12:37.830+00:00	/Volumes/workspace/sd_schema/datas...	null
Male	Bronze	555-980-4337	null	null	2025-11-18T05:13:54.112+00:00	/Volumes/workspace/sd_schema/datas...	822
Female	Silver	555-916-4679	null	null	2025-11-18T05:13:54.112+00:00	/Volumes/workspace/sd_schema/datas...	711
Male	Gold	555-621-5430	null	null	2025-11-18T05:13:54.112+00:00	/Volumes/workspace/sd_schema/datas...	610
Female	Bronze	555-959-9638	null	null	2025-11-18T05:13:54.112+00:00	/Volumes/workspace/sd_schema/datas...	589
Male	Platinum	555-116-5138	null	null	2025-11-18T05:13:54.112+00:00	/Volumes/workspace/sd_schema/datas...	552
Female	Platinum	555-999-9453	null	null	2025-11-18T05:13:54.112+00:00	/Volumes/workspace/sd_schema/datas...	510
Male	Bronze	555-416-7540	null	null	2025-11-18T05:13:54.112+00:00	/Volumes/workspace/sd_schema/datas...	712
Female	Gold	555-640-2842	null	null	2025-11-18T05:13:54.112+00:00	/Volumes/workspace/sd_schema/datas...	801
Male	Gold	555-795-7023	null	null	2025-11-18T05:13:54.112+00:00	/Volumes/workspace/sd_schema/datas...	520
Female	Silver	555-298-1940	null	null	2025-11-18T05:13:54.112+00:00	/Volumes/workspace/sd_schema/datas...	482

Bronze Layer Implementation (AddNewColumns Data)

```

map_keys(col("rescued_data_json_to_map"))
).alias("rescued_key")
).distinct()

# Collect keys as a list (only if df is not streaming)
# If streaming, you must provide the list of possible keys another way
new_keys = [row["rescued_key"] for row in df_keys.collect()] if not df.isStreaming else []

```

Last update: Nov 17, 2025, 11:53:33 PM

	SignUpDate	_rescued_data	ingestion_datetime	source_filename	Age	Gender	LoyaltyStatus	CreditScore
1	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	53	Female	Bronze	822
2	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	40	Male	Silver	711
3	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	18	Female	Gold	610
4	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	49	Male	Bronze	589
5	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	57	Female	Platinum	552
6	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	66	Male	Platinum	510
7	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	41	Female	Bronze	712
8	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	37	Male	Gold	801
9	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	67	Female	Gold	520
10	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	49	Male	Silver	482
11	2024-04-10	null	2025-11-18T04:51:04.380+00:00	/Volumes/workspace/sd_schema/datastore/c...	49	Male	Platinum	null
12	2024-09-11	null	2025-11-18T04:51:04.380+00:00	/Volumes/workspace/sd_schema/datastore/c...	60	Female	Gold	null
13	2024-02-18	null	2025-11-18T04:51:04.380+00:00	/Volumes/workspace/sd_schema/datastore/c...	22	Male	Bronze	null

Silver Layer Implementation (AddNewColumns Data)

```

# Libraries management
from pl import cust_silver_sd_addnew as pl
from pl import Catalog, workspace
from pl import Schema

```

Last update: Nov 17, 2025, 11:53:33 PM

	SignUpDate	_rescued_data	ingestion_datetime	source_filename	Age	Gender	LoyaltyStatus	CreditScore
1	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	53	Female	Bronze	822
2	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	40	Male	Silver	711
3	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	18	Female	Gold	610
4	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	49	Male	Bronze	589
5	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	57	Female	Platinum	552
6	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	66	Male	Platinum	510
7	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	41	Female	Bronze	712
8	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	37	Male	Gold	801
9	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	67	Female	Gold	520
10	null	null	2025-11-18T04:52:54.620+00:00	/Volumes/workspace/sd_schema/datastore/c...	49	Male	Silver	482
11	2024-04-10	null	2025-11-18T04:51:04.380+00:00	/Volumes/workspace/sd_schema/datastore/c...	49	Male	Platinum	null
12	2024-09-11	null	2025-11-18T04:51:04.380+00:00	/Volumes/workspace/sd_schema/datastore/c...	60	Female	Gold	null
13	2024-02-18	null	2025-11-18T04:51:04.380+00:00	/Volumes/workspace/sd_schema/datastore/c...	22	Male	Bronze	null

Part 1: Workshop

1.1 Understanding Schema Evolution

Schema evolution is a critical capability in modern data engineering that allows data pipelines to adapt to changing data structures without breaking. In real-world scenarios, source systems frequently add new fields,

modify data types, or restructure their output formats. A robust data pipeline must handle these changes gracefully.

Databricks provides three schema evolution modes through the `cloudFiles.schemaEvolutionMode` option:

1. **failOnNewColumns** - Strict mode that rejects any schema changes and stops the pipeline
2. **rescue** - Captures unexpected fields in a special `_rescued_data` column as JSON
3. **addNewColumns** - Automatically adds new columns to the table schema (default when no schema is provided)

1.2 Implementation with "rescue" Mode

Overview

The rescue mode implementation maintains a strict initial schema and captures any unexpected fields in a special `_rescued_data` column. This approach provides complete visibility into schema changes while maintaining strict data quality standards.

Bronze Layer Configuration

The bronze layer was configured to read streaming JSON files from the volume path using Auto Loader (`cloudFiles`). The key configuration parameter `cloudFiles.schemaEvolutionMode` was set to "rescue", which means any fields not present in the initial schema would be captured in the `_rescued_data` column rather than being added as new columns.

Additional metadata columns were added:

- **ingestion_datetime**: Timestamp of when the record was ingested
- **source_filename**: The file path from which the record originated

This metadata is crucial for tracking data lineage and troubleshooting data quality issues.

Silver Layer Configuration

The silver layer applied data quality expectations using DLT's `expect_all_or_drop` feature. Two critical expectations were implemented:

1. **no_rescued_data**: This expectation enforces that `_rescued_data IS NULL`, meaning no unexpected fields should be present. This is a strict quality check that will drop any rows containing rescued data.
2. **valid_id**: This expectation validates that `CustomerID IS NOT NULL`, ensuring every record has a valid identifier.

The combination of these expectations creates a strict data quality boundary - only records that match the expected schema and have valid identifiers will pass through to the silver layer.

Processing Rescued Data

Two custom functions were implemented to handle rescued data:

Function 1: `process_rescue_data_new_fields()`

This function extracts new fields from the `_rescued_data` JSON column and adds them as proper columns to the dataframe. It converts the JSON string into a map structure, then iterates through expected fields to extract values. If a field already exists in the dataframe but has a value in `rescued_data`, it uses the rescued value (handling the case where a field was added after initial processing).

Function 2: `process_rescue_data_datatype_change()`

This function handles data type changes for existing columns. When a column's data type changes in the source (for example, `SignupDate` changing from String to Date), the new values appear in `_rescued_data`. This function checks for each column whether it has a value in `rescued_data`, and if so, casts it to the appropriate target data type. After processing, it sets `_rescued_data` back to NULL to indicate the data has been properly handled.

Workflow with Rescue Mode

The typical workflow when using rescue mode involves:

1. **Initial Load:** Upload customer_data_1.json with base schema - pipeline succeeds
2. **Schema Change Detection:** Upload customer_data_2.json with new fields (Age, Gender, LoyaltyStatus) - new fields go into `_rescued_data`
3. **Pipeline Failure:** The expectation `_rescued_data IS NULL` causes the pipeline to fail because rescued data is present
4. **Full Table Refresh:** Must perform a "Full Table Refresh" to restructure the tables
5. **Code Update:** Uncomment the processing functions in the silver layer to handle rescued data
6. **Successful Processing:** After refresh, the functions extract fields from `_rescued_data` and the pipeline succeeds

1.3 Implementation with "addNewColumns" Mode

Overview

The addNewColumns mode provides automatic schema evolution by adding new columns to the table schema as they appear in the source data. This mode is more flexible and requires less manual intervention compared to rescue mode.

Bronze Layer Configuration

The bronze layer configuration was similar to rescue mode, but with `cloudFiles.schemaEvolutionMode` set to "addNewColumns". This setting tells Auto Loader to automatically add any new columns it discovers to the table schema. The same metadata columns (ingestion_datetime and source_filename) were added for consistency.

Silver Layer Configuration

The silver layer with addNewColumns mode uses a simpler approach. Instead of strict expectations about rescued data, it only validates that `CustomerID IS NOT NULL`. The transformation logic includes conditional data type casting:

Conditional Type Casting Logic:

The silver layer checks for the presence of specific columns before applying type conversions. This conditional approach is necessary because:

- Not all source files may contain all columns
- Schema evolves over time as new files are uploaded
- Applying type casting to non-existent columns would cause errors

The implemented type conversions were:

- **SignupDate:** Cast from String to DateType for proper date operations
- **Age:** Cast from String/inferred type to IntegerType for numeric operations
- **CreditScore:** Cast from String/inferred type to IntegerType (this was the bug fix)

Workflow with AddNewColumns Mode

The workflow is much simpler:

1. **Initial Load:** Upload customer_data_1.json - schema established with 6 base columns
2. **Automatic Evolution:** Upload customer_data_2.json - Age, Gender, LoyaltyStatus automatically added as columns
3. **Continued Processing:** Upload customer_data_3.json - no schema changes, data flows normally
4. **Final Evolution:** Upload customer_data_4.json - CreditScore added (after fix)

No pipeline failures occur, and no full table refresh is needed for new columns (only for data type changes of existing columns).

1.4 Detailed Observations and Behavioral Differences

Observation 1: Pipeline Stability

Rescue Mode:

When customer_data_2.json was uploaded, the pipeline immediately failed because the expectation `_rescued_data IS NULL` was violated. The new fields (Age, Gender, LoyaltyStatus) were captured in the `_rescued_data` column, causing all rows from this file to be dropped. This failure is by design - it alerts data engineers that the schema has changed and requires attention.

The failure provides strong governance but disrupts data flow. In a production environment, this means data stops flowing until the issue is addressed, which could impact downstream processes and analytics.

AddNewColumns Mode:

When customer_data_2.json was uploaded, the pipeline continued running without any failures. The new columns were automatically added to both bronze and silver tables. Data continued to flow seamlessly, with NULL values appearing in the new columns for records from earlier files.

This automatic handling provides operational continuity but may allow unexpected schema changes to propagate through the pipeline without immediate visibility.

Observation 2: Schema Change Handling

Rescue Mode:

All schema changes are captured in `_rescued_data` as a JSON string. This creates a complete audit trail of what fields were unexpected. However, this data is not immediately usable - it requires custom parsing logic to extract and properly type the fields.

The format in `_rescued_data` looks like:

```
{"Age": "26", "Gender": "Female", "LoyaltyStatus": "Platinum"}
```

To make this data usable, we need to:

1. Parse the JSON string into a map
2. Extract each field individually
3. Cast to appropriate data types
4. Handle cases where both the regular column and rescued_data contain values

AddNewColumns Mode:

Schema changes result in actual new columns being added to the table. Each new field becomes a first-class column with proper typing (based on Auto Loader's inference). For existing records that don't have these fields, the values are NULL.

This creates a sparse table structure where newer columns have many NULL values for older records, but the data is immediately queryable without any parsing.

Observation 3: Data Type Changes

Rescue Mode:

When a column's data type needs to change (like SignupDate going from String to Date), rescue mode can capture the new values in `_rescued_data`. The custom function `process_rescue_data_datatype_change()` handles this by:

1. Checking if the column exists in `_rescued_data`
2. Casting the rescued value to the target type
3. Using the rescued value if present, otherwise using the original column value

This allows for gradual data type migration, but requires a full table refresh to apply the change to the entire table.

AddNewColumns Mode:

Data type changes cannot be handled automatically with addNewColumns mode. If a column needs to change type, you must:

1. Perform a full table refresh
2. Update the silver layer transformation logic to cast the column

3. Reprocess all historical data with the new type

The conditional casting approach in the silver layer (checking `if "Age" in df.columns` before casting) ensures the code doesn't fail when processing older data that might not have the column yet.

Observation 4: Code Complexity

Rescue Mode:

The implementation requires significantly more code:

- Two custom functions (200+ lines) to process rescued data
- Complex JSON parsing logic
- Map extraction and type casting
- Careful handling of edge cases where both regular column and rescued_data contain values

This complexity increases maintenance burden and the potential for bugs. However, it also provides fine-grained control over how schema changes are handled.

AddNewColumns Mode:

The implementation is straightforward:

- Simple conditional checks for column existence
- Direct type casting without parsing
- Minimal custom logic (about 20 lines)

The simplicity reduces maintenance overhead and makes the code easier to understand and debug.

Observation 5: Full Refresh Requirements

Rescue Mode:

Full table refresh is required whenever:

- Schema changes are detected (to restructure tables to accommodate rescued data)
- Data type changes occur
- Processing logic is updated

The refresh requirement means:

- Entire bronze and silver tables are reprocessed
- Downtime during refresh operation
- Computational cost of reprocessing all historical data

AddNewColumns Mode:

Full table refresh is only required when:

- Data type changes occur for existing columns
- Processing logic fundamentally changes

New columns are added automatically without refresh, which means:

- No downtime for additive schema changes
- Lower computational cost
- Faster adaptation to schema changes

However, when a refresh IS needed (for type changes), both modes require the same level of effort.

Observation 6: Data Quality Control

Rescue Mode:

Provides strongest data quality control:

- Explicit approval needed for schema changes (via code updates)

- Complete visibility into unexpected fields
- Ability to reject or quarantine non-conforming data
- Audit trail of all schema deviations

The expectation `_rescued_data IS NULL` acts as a circuit breaker that stops problematic data from flowing through the pipeline.

AddNewColumns Mode:

Provides weaker data quality control:

- Schema changes are accepted automatically
- Unexpected columns are added without explicit approval
- No built-in mechanism to alert on schema changes
- Relies on downstream validation to catch issues

However, the expectation `CustomerID IS NOT NULL` still provides basic data quality validation.

Observation 7: Storage and Performance

Rescue Mode:

- Bronze table has fixed column count with `_rescued_data` containing JSON
- JSON storage is less efficient than native column storage
- Queries against rescued data require JSON parsing functions
- Silver table structure changes after processing rescued data

Storage pattern:

```
Initial: 8 columns (6 business + 2 metadata)
After schema change: 8 columns (6 business + _rescued_data + 2 metadata)
After processing: 11 columns (9 business + _rescued_data [NULL] + 2 metadata)
```

AddNewColumns Mode:

- Column count increases with each schema change
- Native column storage is more efficient
- Direct column queries without parsing
- Sparse table structure with many NULLs

Storage pattern:

```
Initial: 8 columns (6 business + 2 metadata)
After schema change: 11 columns (9 business + 2 metadata)
After another change: 12 columns (10 business + 2 metadata)
```

Observation 8: Use Case Suitability

Rescue Mode is ideal for:

- Highly regulated industries (finance, healthcare) requiring strict data governance
- Scenarios where schema changes must be reviewed and approved
- Data contracts with external providers where schemas should be stable
- Audit and compliance requirements needing complete change tracking
- Production systems where unexpected schema changes indicate upstream problems

AddNewColumns Mode is ideal for:

- Agile development environments with frequent iterations
- IoT or sensor data where new metrics are regularly added
- Semi-structured data sources with variable schemas

- Rapid prototyping and experimentation
- Data lakes ingesting data from multiple evolving sources

1.5 File-by-File Schema Evolution Timeline

customer_data_1.json (Base Schema)

This file established the initial schema with 6 business columns:

- CustomerID: Unique identifier for each customer
- FullName: Customer's full name
- Email: Contact email address
- PhoneNumber: Contact phone number
- City: Customer's city of residence
- SignupDate: Date when customer signed up (String format)

Plus 2 metadata columns:

- ingestion_datetime: System timestamp
- source_filename: Source file path

Total columns in Bronze: 8 Total rows: 5 customers

This file processed successfully in both rescue and addNewColumns modes as it defined the baseline schema.

customer_data_2.json (First Schema Evolution)

This file introduced 3 new fields while retaining all original fields:

- Age: Customer's age (Integer)
- Gender: Customer's gender (String: Male/Female/Non-binary)
- LoyaltyStatus: Customer tier (String: Bronze/Silver/Gold/Platinum)

Additionally, some data values changed for existing customers (C001-C004), demonstrating that schema evolution often accompanies data updates.

New columns added: 3 Total columns in Bronze (addNewColumns): 11 Total rows added: 8 (includes 3 new customers C005-C007, C008)

Rescue Mode Behavior:

- New fields captured in `_rescued_data : {"Age": "26", "Gender": "Female", "LoyaltyStatus": "Platinum"}`
- Pipeline failed due to expectation violation
- Required full table refresh and code update to process

AddNewColumns Mode Behavior:

- Age, Gender, LoyaltyStatus automatically added as columns
- Old records (from customer_data_1.json) show NULL for these columns
- New records have values populated
- Pipeline continued without interruption

customer_data_3.json (Data Updates, No Schema Changes)

This file maintained the same schema as customer_data_2.json but with significant data updates:

- Updated values for existing customers (C001-C009)
- Added 1 new customer (C011)
- Demonstrated that not every file upload needs schema changes

Schema changes: None **Total columns remain: 11 (in addNewColumns mode)** **Total rows added: 9**

Both Modes Behavior:

- Processed seamlessly in both modes
- No schema evolution logic triggered
- Data simply appended/updated based on streaming semantics
- Demonstrated stable operation when schema doesn't change

This file validated that both approaches handle normal data ingestion efficiently when schema is stable.

customer_data_4.json (Second Schema Evolution + Field Removal)

This file introduced interesting changes:

- Added new field: CreditScore (Integer) - customer's credit score
- Removed field: SignupDate no longer present in new records
- Updated values for all existing customers (C001-C010)

New columns added: 1 (CreditScore) **Total columns in Bronze (addNewColumns): 12** **Columns removed from new records: 1 (SignupDate)** **Total rows added: 10**

Critical Observation about Field Removal:

SignupDate was NOT removed as a column from the table - it remained as a column but with NULL values for records from customer_data_4.json. This is expected behavior: schema evolution in Delta Lake is additive only. Once a column exists, it persists in the table schema even if future records don't populate it.

AddNewColumns Mode Behavior (Before Fix):

- CreditScore added to bronze layer successfully
- CreditScore did NOT appear in silver layer (BUG)
- Reason: Silver layer transformation lacked conditional check for CreditScore

AddNewColumns Mode Behavior (After Fix):

- CreditScore added to bronze layer
- CreditScore properly cast to IntegerType and appeared in silver layer
- Fix involved adding the conditional casting logic

Rescue Mode Behavior:

- CreditScore captured in `_rescued_data`
- Required updating the `expected_new_fields` list to include "CreditScore"
- Required full refresh to apply changes
- After processing, CreditScore extracted from rescued data and properly typed

1.6 Comprehensive Comparison Matrix

Criteria	Rescue Mode	AddNewColumns Mode	Automatic Schema Evolution	No - manual intervention required	Yes - fully automatic	Pipeline Failures on Schema Change	Yes - intentional failure for governance	No - continues seamlessly	New Field Handling	Stored in <code>_rescued_data</code> as JSON	Added as new native columns	Data Type Change	Handling	Custom function required	Custom function required	Full Table Refresh	Frequency	Every schema change	Only for data type changes	Code Complexity	High (200+ lines of custom logic)	Low (20 lines of conditional casts)	Data Quality Enforcement	Very strict - circuit breaker pattern	Moderate - basic validation only	Audit Trail	Complete - all changes	in <code>_rescued_data</code>	None - changes happen silently	Query Performance	Slower - requires JSON parsing	Faster - native column access	Storage Efficiency	Lower - JSON string overhead	Higher - native column storage	Operational Overhead	High - requires monitoring and intervention	Low - mostly self-managing	Schema Governance	Strong - explicit approval needed	Weak - automatic acceptance	Development Agility	Slower - requires code updates	Faster - automatic adaptation	Production Stability	Can cause outages during schema changes
----------	-------------	--------------------	----------------------------	-----------------------------------	-----------------------	------------------------------------	--	---------------------------	--------------------	--	-----------------------------	------------------	----------	--------------------------	--------------------------	--------------------	-----------	---------------------	----------------------------	-----------------	-----------------------------------	-------------------------------------	--------------------------	---------------------------------------	----------------------------------	-------------	------------------------	-------------------------------	--------------------------------	-------------------	--------------------------------	-------------------------------	--------------------	------------------------------	--------------------------------	----------------------	---	----------------------------	-------------------	-----------------------------------	-----------------------------	---------------------	--------------------------------	-------------------------------	----------------------	---

Changes	Maintains continuous operation	Debugging Complexity	Higher - more code paths to trace	Lower - straightforward logic
Testing Requirements	Extensive - test rescue data processing	Moderate - test type conversions		
Documentation Needs	High - complex logic to document	Low - simple logic to document		
Team Skill Requirements	Advanced - JSON processing, complex PySpark	Intermediate - basic PySpark	Cost (Compute)	Higher - more processing for parsing
Team Size	Larger teams with dedicated data engineers	Smaller teams, self-service analytics		

1.7 Key Insights and Recommendations

Insight 1: No Universal "Best" Approach

Neither mode is universally superior - the choice depends on organizational context:

Choose Rescue Mode when:

- Operating in regulated industries (financial services, healthcare, government)
- Data contracts are formal agreements requiring change management
- Schema changes are rare and should be exceptional events
- Team has advanced PySpark skills
- Audit and compliance requirements are stringent
- Budget allows for dedicated data engineering resources

Choose AddNewColumns Mode when:

- Operating in agile, fast-paced environments
- Source systems evolve rapidly
- Team prefers simplicity over strict control
- Self-service analytics is a priority
- Time-to-insight is critical
- Budget favors minimal operational overhead

Insight 2: Hybrid Approaches Are Possible

Organizations don't have to choose one mode for all pipelines. Consider:

- **Tier-based approach:** Use rescue mode for critical tier-1 data, addNewColumns for tier-2/3
- **Environment-based:** Rescue in production, addNewColumns in development
- **Source-based:** Rescue for external vendors, addNewColumns for internal systems
- **Maturity-based:** Start with addNewColumns, graduate to rescue as governance matures

Insight 3: The Real Challenge is Data Type Changes

Both modes struggle with data type changes for existing columns. Neither mode can automatically convert a column from String to Integer without a full table refresh. This is a fundamental limitation of Delta Lake's schema evolution.

Mitigation strategies:

- Define clear data type contracts upfront
- Use String types for flexibility (cast in silver/gold layers)
- Plan for periodic full refreshes during maintenance windows
- Consider creating new columns (e.g., Age_v2) rather than changing types

Insight 4: Metadata is Crucial

Both implementations included `ingestion_datetime` and `source_filename` metadata columns. This proved invaluable for:

- Debugging schema evolution issues
- Understanding data lineage
- Identifying which file introduced which schema changes
- Time-based analysis of data evolution

Recommendation: Always include comprehensive metadata regardless of schema evolution mode chosen.

Insight 5: Testing Schema Evolution is Complex

Schema evolution testing requires:

- Sequential file uploads in specific order
- Validation at each stage
- Checking both success and failure scenarios
- Verifying data type conversions
- Ensuring NULL handling is correct

This makes automated testing challenging. The assignment's sequential file approach (`customer_data_1` → `2` → `3` → `4`) provided a realistic testing scenario.

Part 2: Fixing the issue

2.1 Problem Identification

Issue Description

When `customer_data_4.json` was uploaded, the new field **CreditScore** was successfully appearing in the **bronze layer** table but was **completely missing** from the **silver layer** table.

Impact

- Critical data loss between bronze and silver layers
- Business users unable to access CreditScore for analytics
- Schema evolution not propagating through the pipeline as expected

Initial Verification

Bronze Layer Check:

- CreditScore column: Present
- Data values: 10 non-NULL records with scores ranging from 482 to 822
- Column type: String (inferred by Auto Loader)

Silver Layer Check:

- CreditScore column: Completely absent
- Total columns: 11 (should be 12)
- Data loss confirmed

2.2 Root Cause Analysis

Investigation Steps

Step 1: Bronze Layer Review

Confirmed that bronze layer was correctly configured with `cloudFiles.schemaEvolutionMode = "addNewColumns"`. Auto Loader

successfully detected and added the CreditScore column.

Step 2: Silver Layer Transformation Review

Discovered that the silver layer transformation function only had explicit handling for:

- SignupDate (cast to DateType)
- Age (cast to IntegerType)

Step 3: Identified the Gap

The transformation logic had **NO conditional check** for CreditScore, meaning it was read from bronze but never explicitly processed or passed through to the output.

Root Cause Explanation

The silver layer uses **explicit column transformation**. Only columns that are explicitly processed get propagated to the output dataframe. Since CreditScore was not included in any transformation logic, it was implicitly dropped during the DataFrame operations.

Key Insight: While `addNewColumns` mode automatically adds columns in bronze, it does NOT automatically propagate them through custom transformations in downstream layers. Each transformation layer needs explicit logic to handle new columns.

2.3 The Solution

Fix Applied

Added a conditional block to handle CreditScore in the silver layer transformation:

Logic Added:

- Check if "CreditScore" column exists in the bronze dataframe
- If present, cast it from String to IntegerType
- Include it in the output dataframe

Why This Fix Works

1. **Conditional Check Prevents Errors:** The `if "CreditScore" in df.columns` check ensures the code doesn't fail when processing older files (customer_data_1, 2, 3) that don't contain CreditScore
2. **Explicit Processing Ensures Propagation:** By explicitly casting CreditScore with `withColumn()`, we ensure it's included in the output dataframe written to silver table
3. **Type Casting Provides Data Quality:** Converting to IntegerType ensures CreditScore can be used in mathematical operations and aggregations
4. **Maintains Code Consistency:** Follows the same pattern as existing Age and SignupDate transformations

Before Fix

Silver layer transformation had:

- SignupDate handling
- Age handling
- CreditScore handling (MISSING - causing the bug)

After Fix

Silver layer transformation has:

- SignupDate handling
- Age handling
- CreditScore handling (ADDED - resolves the bug)