databricks

Transform Data with Spark

Module 02

Module Objectives

Transform Data with Spark

1. Extract data from a variety of file formats and data sources using Spark
2. Apply a number of common transformations to clean data using Spark
3. Reshape and manipulate complex data using advanced built-in functions in Spark
4. Leverage UDFs for reusable code and apply best practices for performance in Spark

Module Agenda

Transform Data with Spark

Data Objects in the Lakehouse

DE 2.1 – Querying Files Directly

DE 2.2 – Options for External Sources

DE 2.3L – Extract Data Lab

DE 2.4 – Cleaning Data

DE 2.5 – Complex Transformations

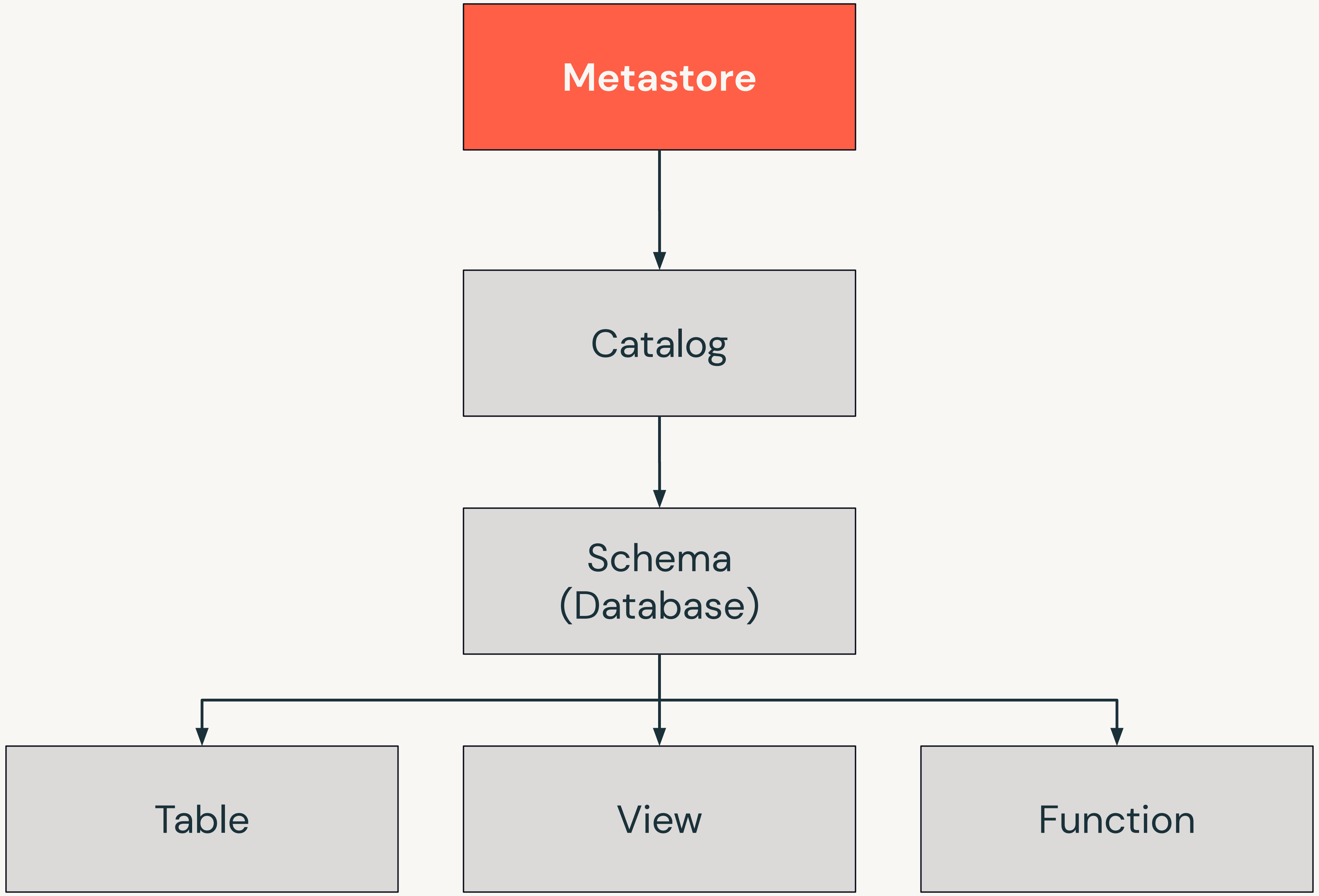
DE 2.6L – Reshape Data Lab

DE 2.7A – SQL UDFs and Control Flow

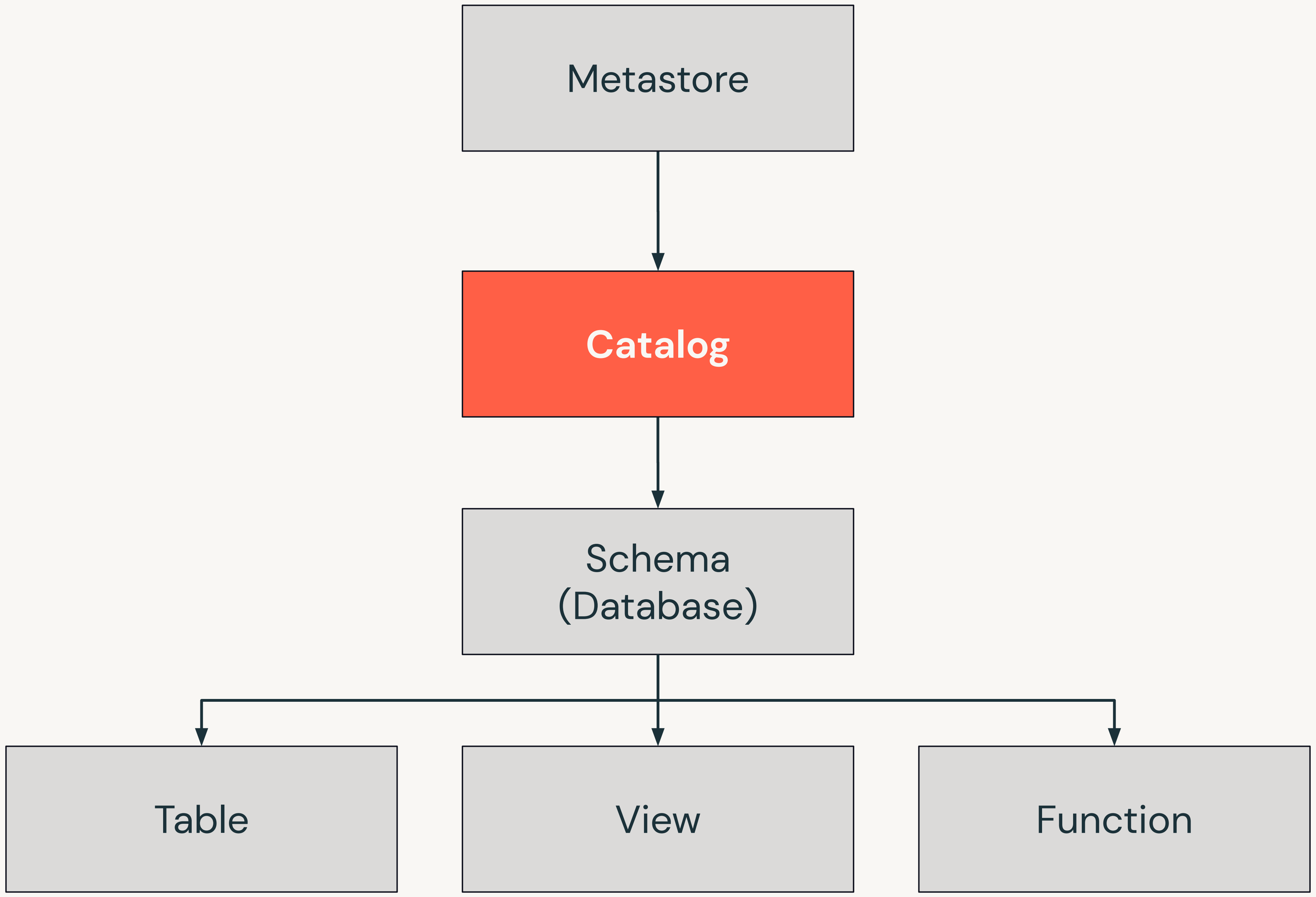
DE 2.7B – Python UDFs

Data Objects in the Lakehouse

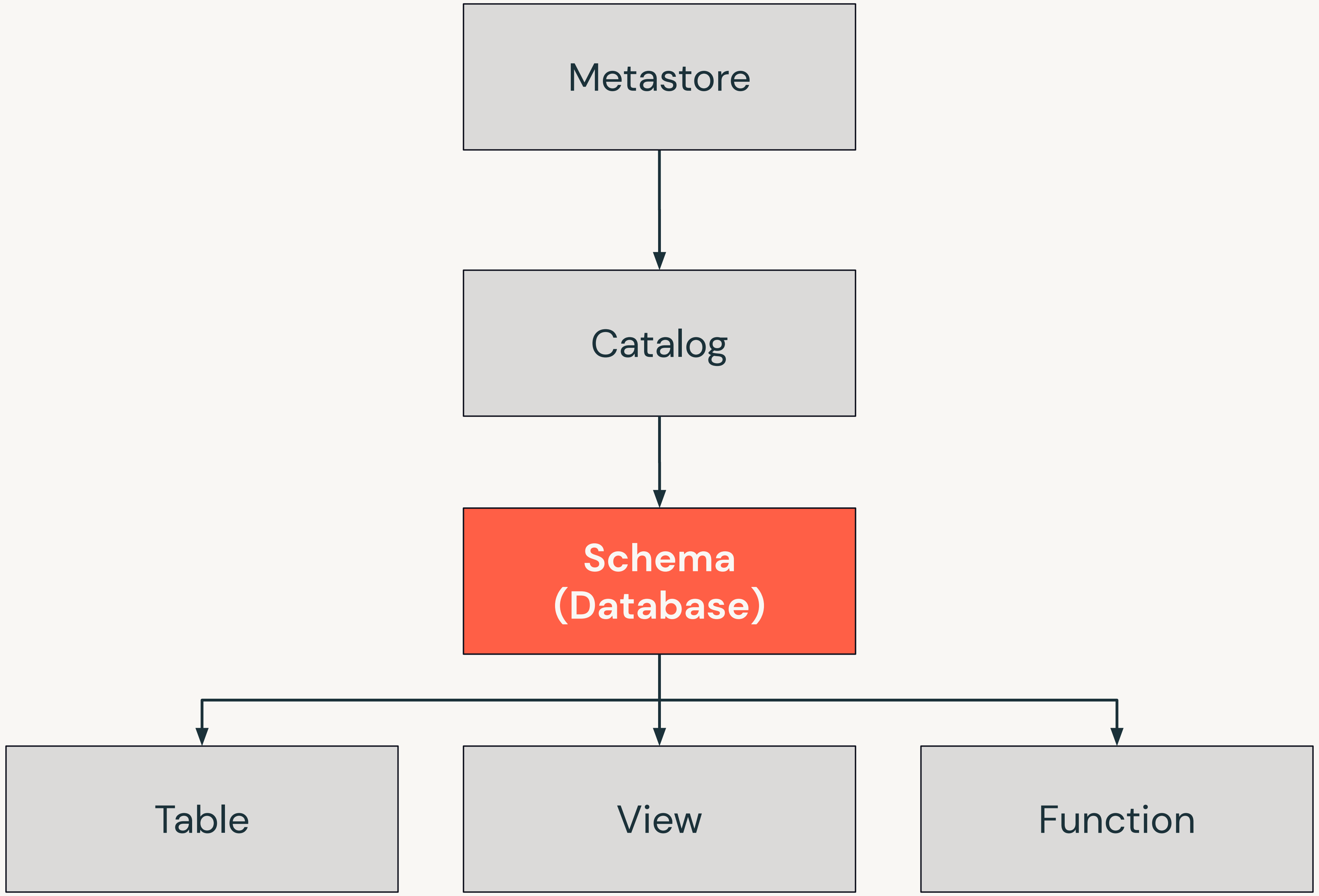
Data objects in the Lakehouse



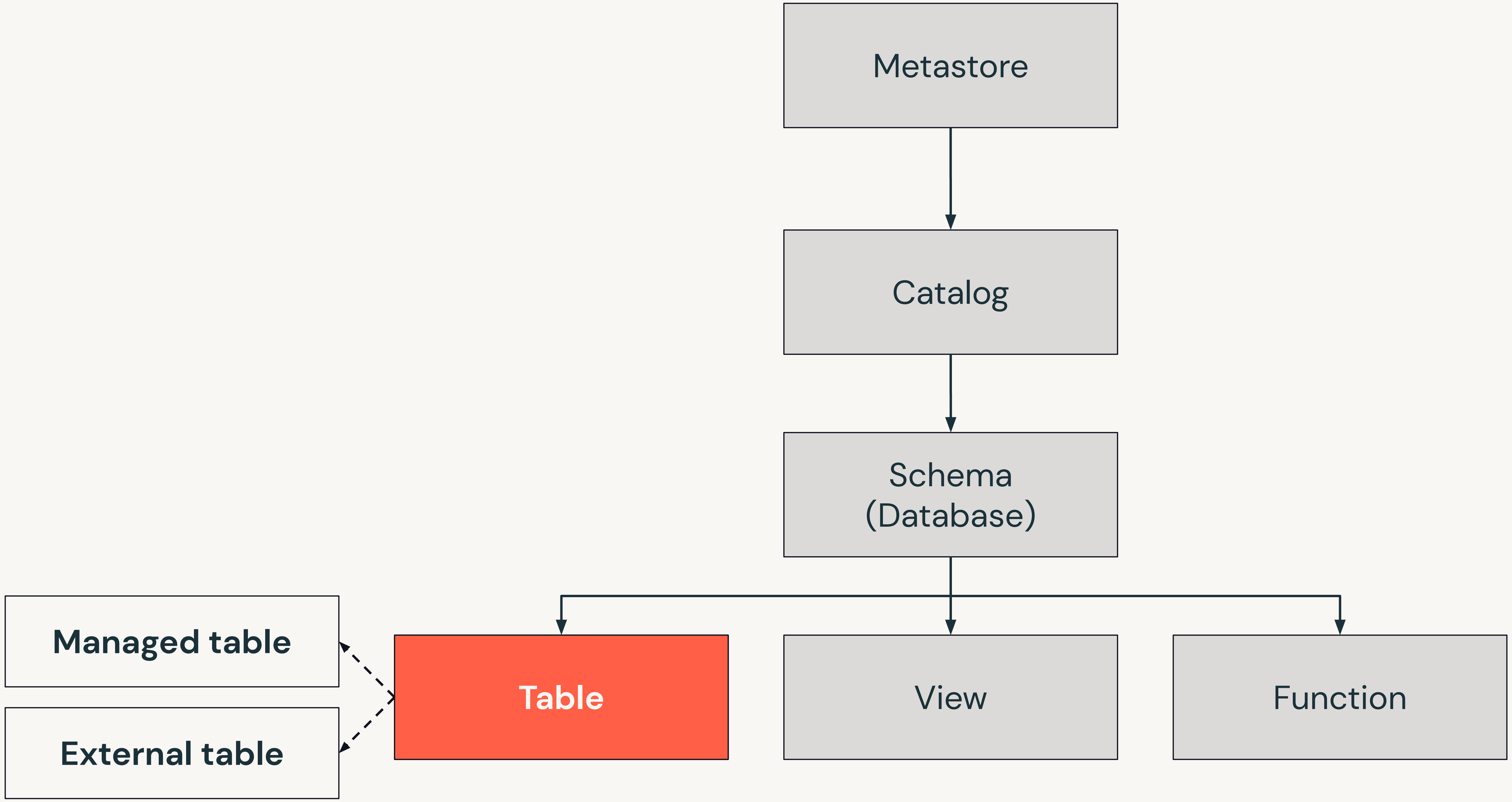
Data objects in the Lakehouse



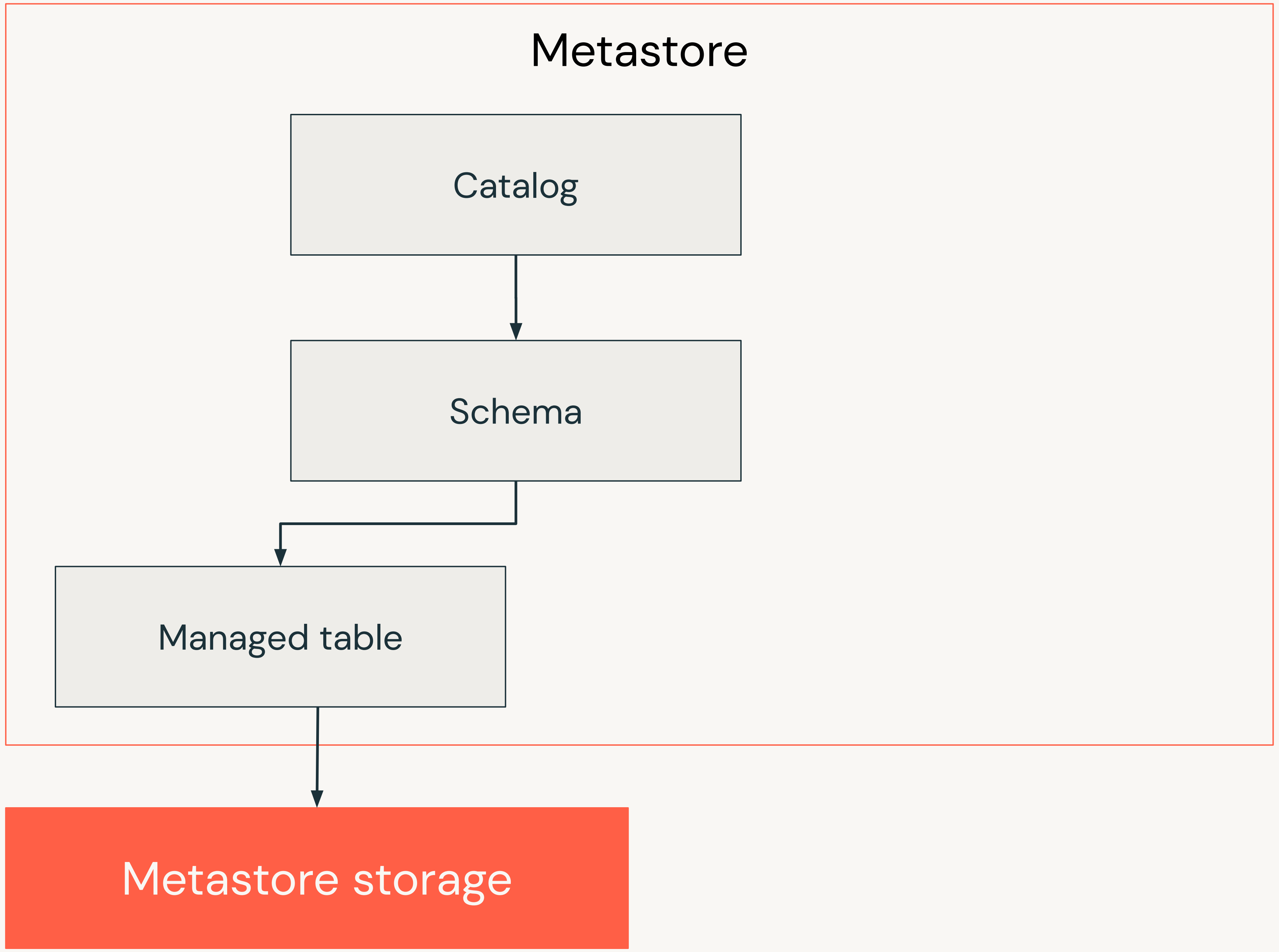
Data objects in the Lakehouse



Data objects in the Lakehouse



Managed Tables



Metastore

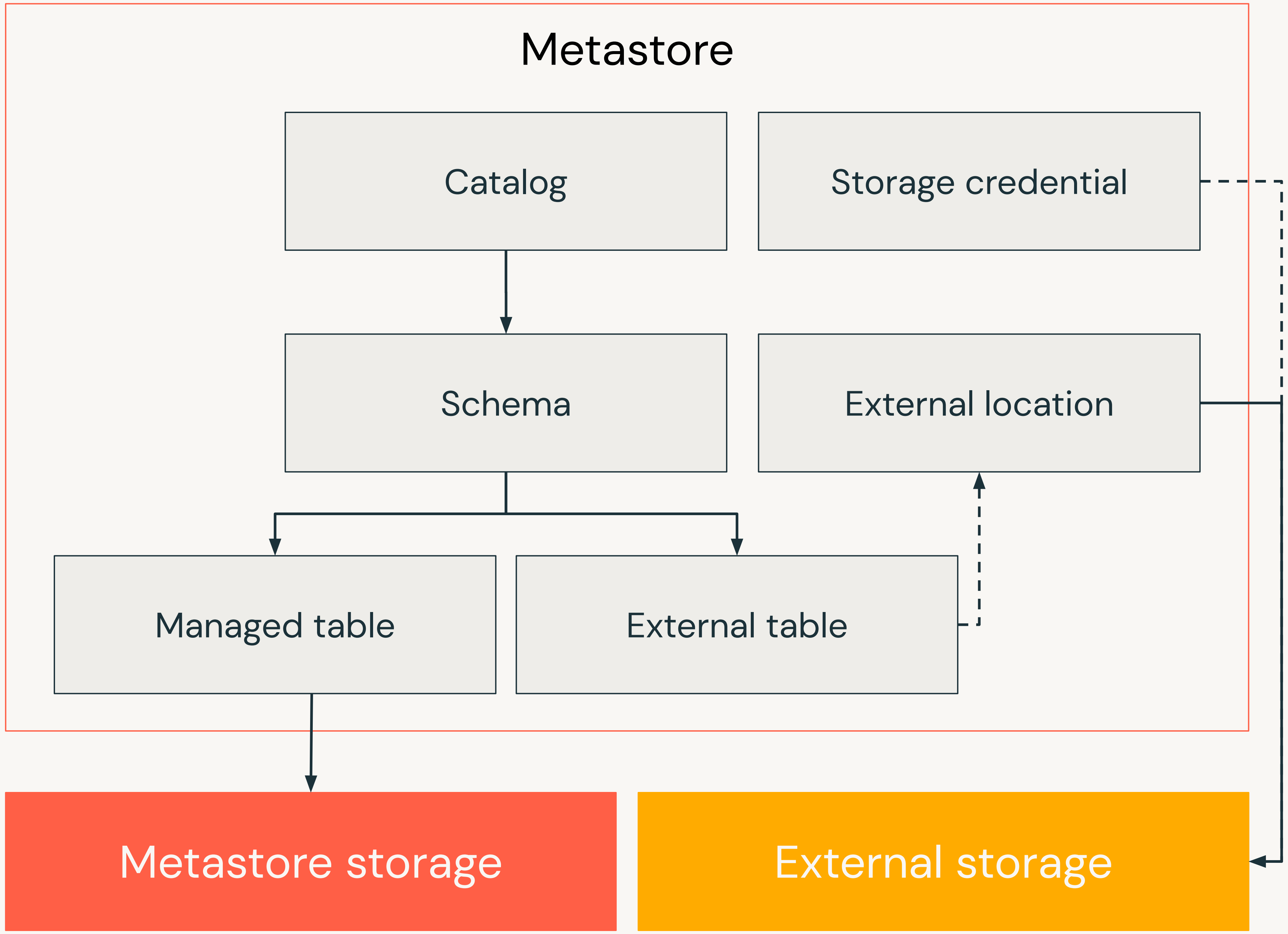
Catalog

Schema

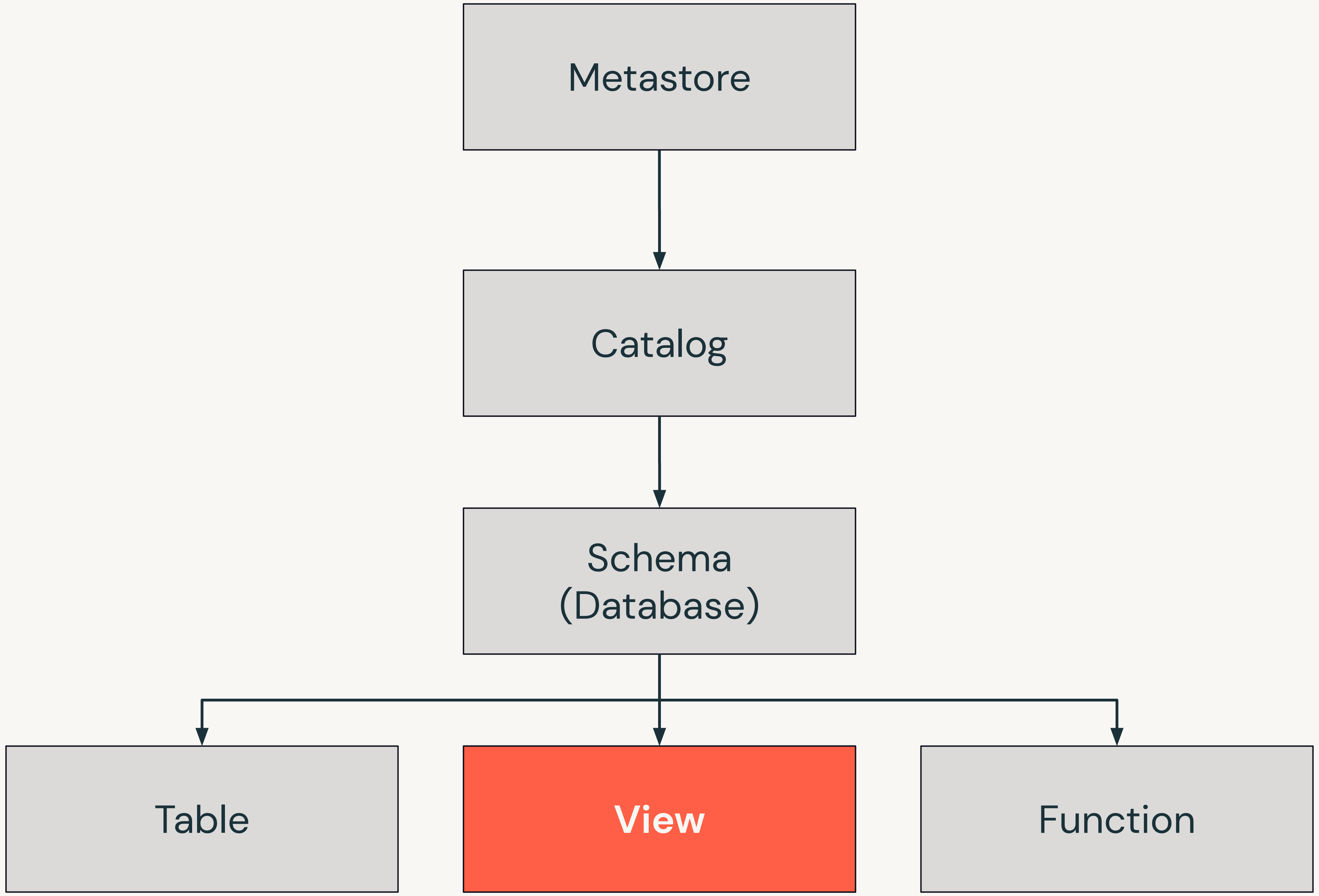
Managed table

Metastore storage

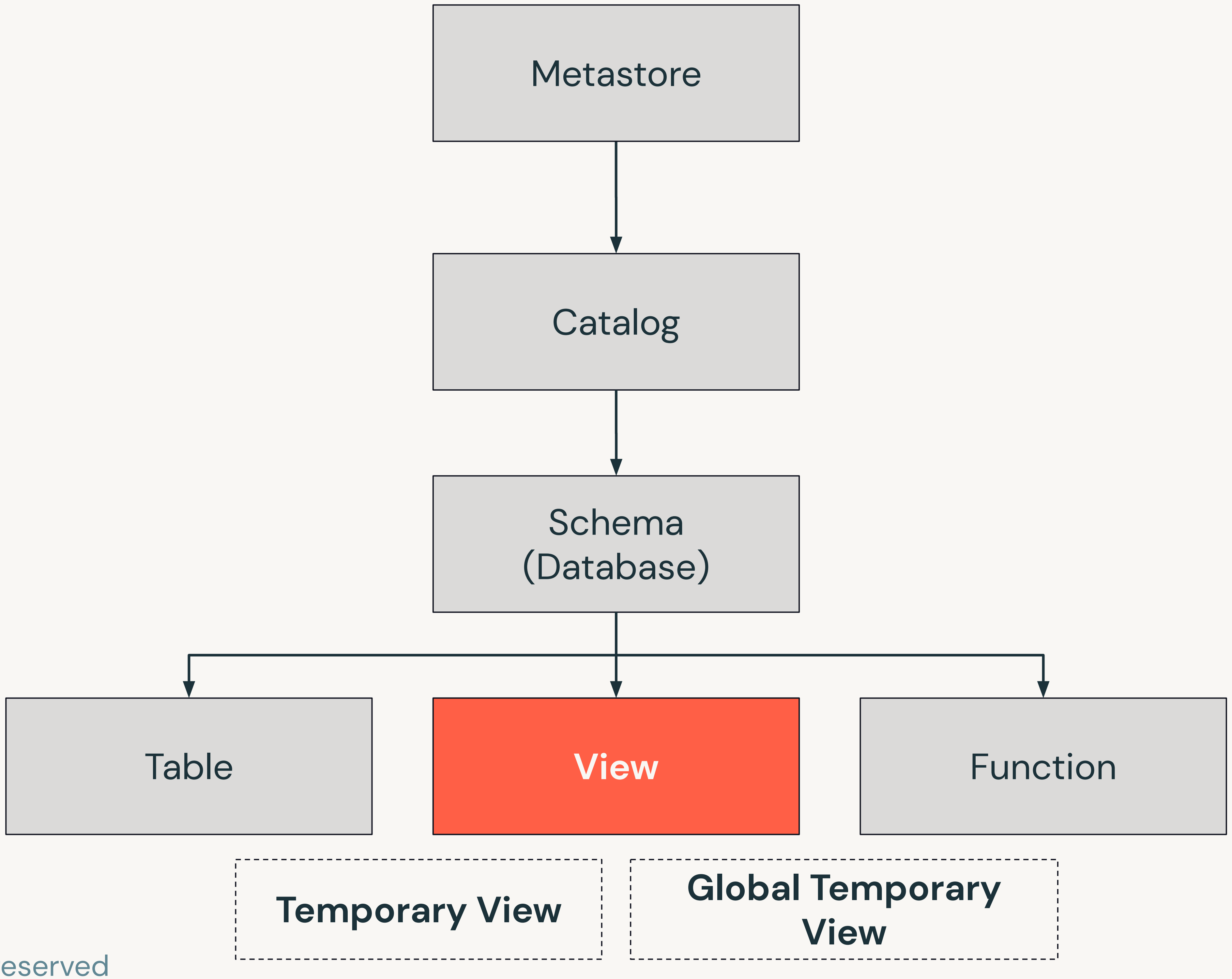
External Tables



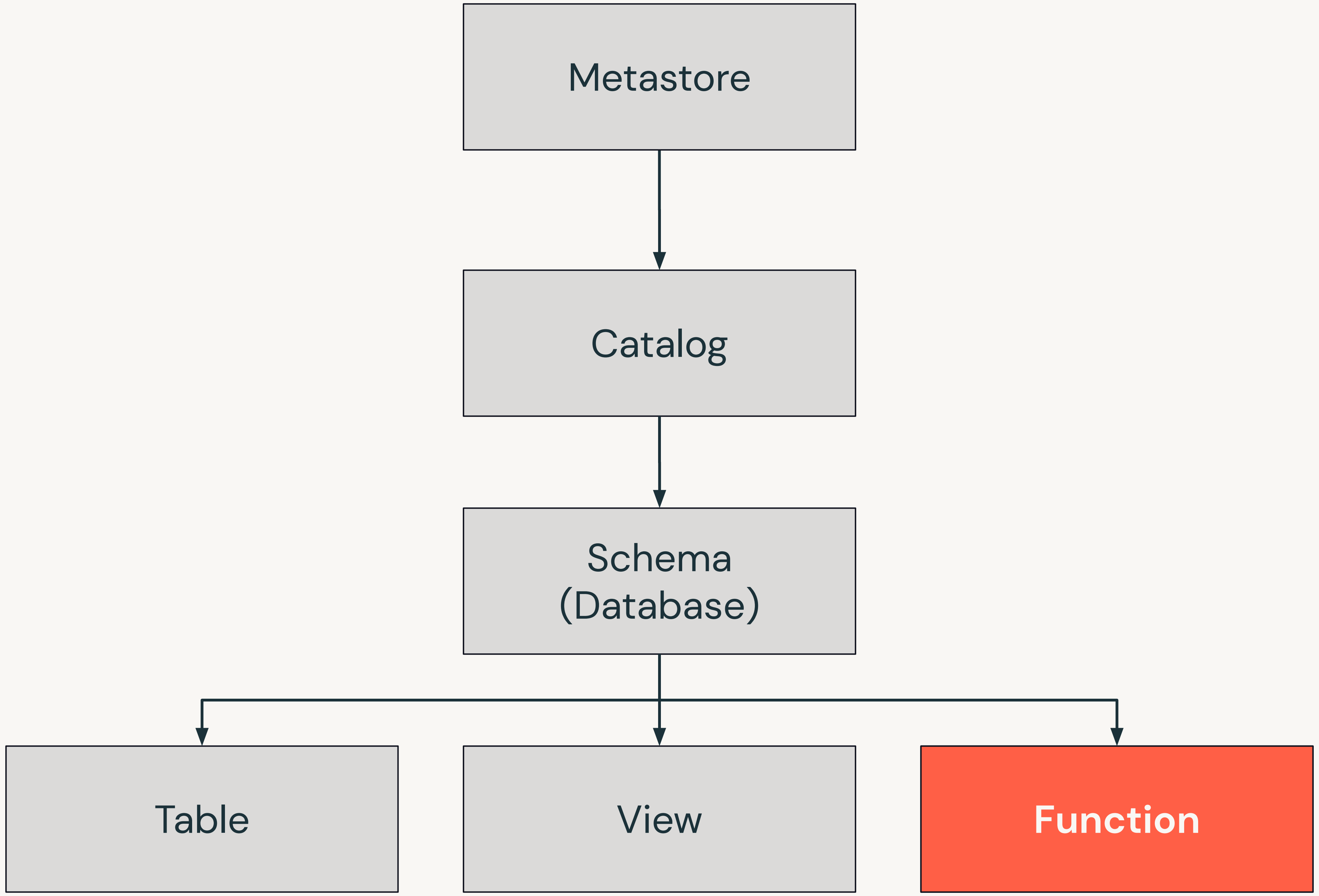
Data objects in the Lakehouse



Data objects in the Lakehouse



Data objects in the Lakehouse



Extracting Data

Query files directly

`SELECT * FROM file_format.`path/to/file``

Files can be queried directly using SQL

- `SELECT * FROM json.`path/to/files/``
- `SELECT * FROM text.`path/to/files/``

Process based on specified file format

- `json` pulls schema from underlying data
- `binaryFile` and `text` file formats have fixed data schemas
  - `text` → string `value` column (row for each line)
  - `binaryFile` → `path, modificationTime, length, content` columns (row for each file)

Configure external tables with read options

`CREATE TABLE USING data_source OPTIONS (...)`

Many data sources require schema declaration and other options to correctly read data

- CSV options for delimiter, header, etc
- JDBC options for url, user, password, etc
  - Note: using the JDBC driver pulls RDBMS tables dynamically for Spark processing

DE 2.1: Querying Files Directly

Use Spark SQL to directly query JSON data files

Leverage `text` and `binaryFile` methods to review raw file contents

DE 2.2: Providing Options for External Sources

Use Spark SQL to configure options for extracting data from external sources

Create tables against external data sources for various file formats

Describe behavior when querying tables defined against external RDBMS sources

DE 2.3L: Extract Data Lab

DE 2.4: Cleaning Data

Summarize datasets and describe NULL behaviors

Retrieve and removing Duplicates

Validate datasets for expected counts, missing values, and duplicate records

Apply `date_format` and `regexp_extract` to clean and transform data

Complex Transformations

Interact with Nested Data

Use built-in syntax to traverse nested data with Spark SQL

Use “ : ” (colon) syntax in queries to access subfields in JSON strings

```
SELECT value:device, value:geo ...
```

Use “ . ” (dot) syntax in queries to access subfields in STRUCT types

```
SELECT value.device, value.geo ...
```

Complex Types

Nested data types storing multiple values

- **Array**: arbitrary number of elements of same data type
- **Map**: set of key-value pairs
- **Struct**: ordered (fixed) collection of column(s) and any data type

Example table with complex types

```
CREATE TABLE employees (name STRING, salary FLOAT,
   subordinates ARRAY<STRING>,
   deductions   MAP<STRING, FLOAT>,
   address      STRUCT<street:STRING,city:STRING,state:STRING, zip:INT>)
```

DE 2.5: Complex Transformations

Use `:` and `.` syntax to traverse nested data in strings and structs

Use `.*` syntax to flatten and query struct types

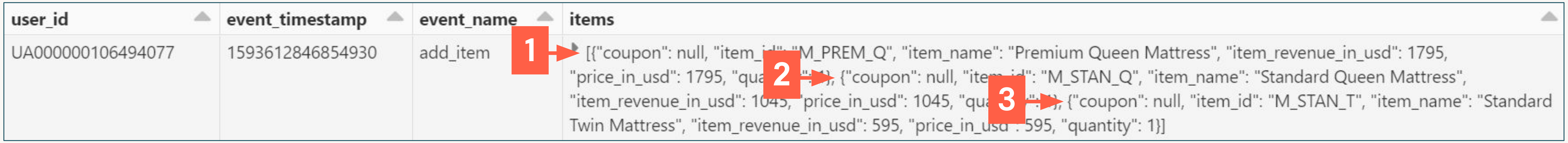
Parse JSON string fields

Flatten/unpack arrays and structs

explode lab

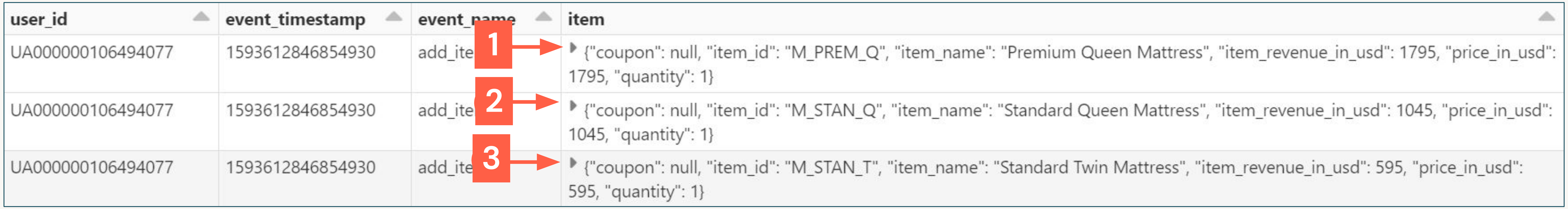
explode outputs the elements of an array field into a separate row for each element

```
SELECT
   user_id, event_timestamp, event_name,
   explode(items) AS item
FROM events
```



user_id	event_timestamp	event_name	items
UA000000106494077	1593612846854930	add_item	[{"coupon": null, "item_id": "M_PREM_Q", "item_name": "Premium Queen Mattress", "item_revenue_in_usd": 1795, "price_in_usd": 1795, "quantity": 1}, {"coupon": null, "item_id": "M_STAN_Q", "item_name": "Standard Queen Mattress", "item_revenue_in_usd": 1045, "price_in_usd": 1045, "quantity": 1}, {"coupon": null, "item_id": "M_STAN_T", "item_name": "Standard Twin Mattress", "item_revenue_in_usd": 595, "price_in_usd": 595, "quantity": 1}]

Each item in the `items` array above is exploded into its own row, resulting in the 3 rows below



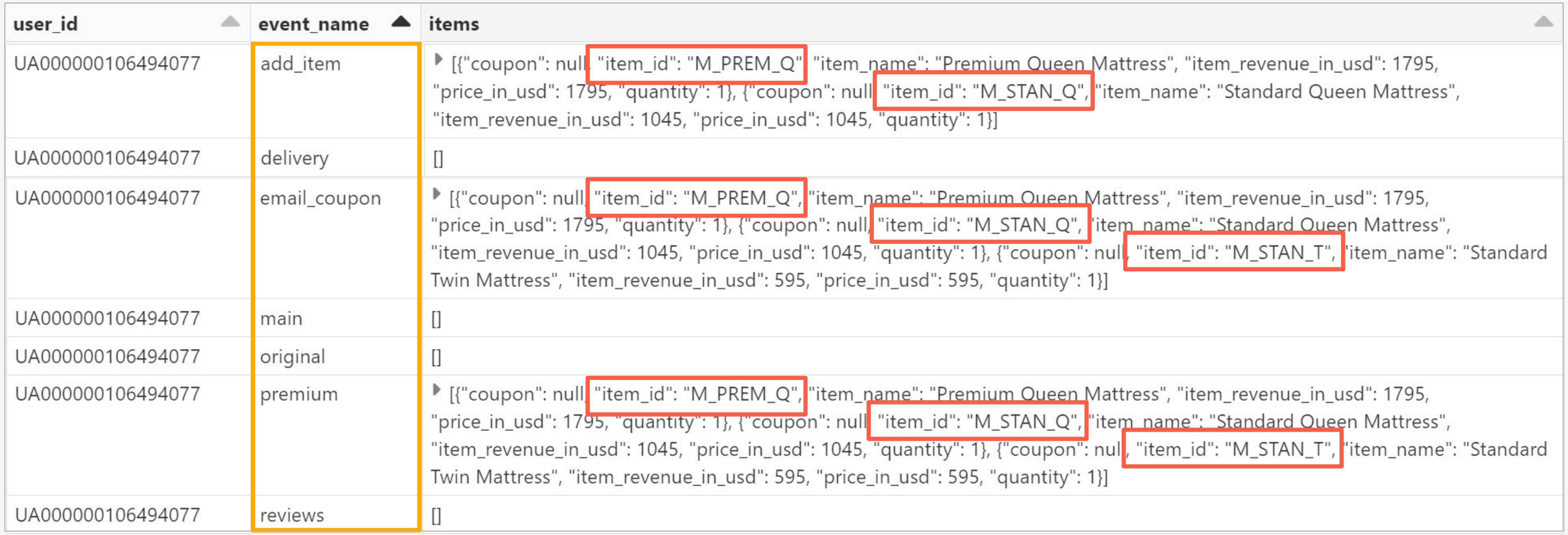
user_id	event_timestamp	event_name	item
UA000000106494077	1593612846854930	add_item	{"coupon": null, "item_id": "M_PREM_Q", "item_name": "Premium Queen Mattress", "item_revenue_in_usd": 1795, "price_in_usd": 1795, "quantity": 1}
UA000000106494077	1593612846854930	add_item	{"coupon": null, "item_id": "M_STAN_Q", "item_name": "Standard Queen Mattress", "item_revenue_in_usd": 1045, "price_in_usd": 1045, "quantity": 1}
UA000000106494077	1593612846854930	add_item	{"coupon": null, "item_id": "M_STAN_T", "item_name": "Standard Twin Mattress", "item_revenue_in_usd": 595, "price_in_usd": 595, "quantity": 1}

flatten lab

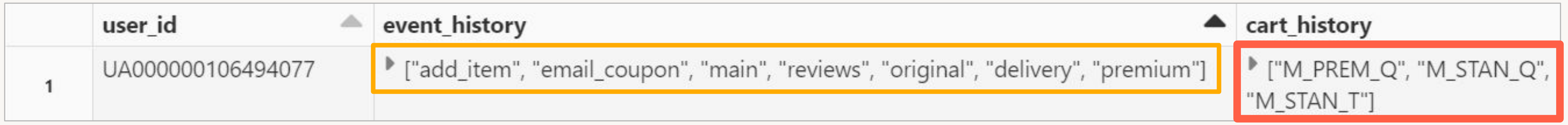
collect_set returns an array of unique values from a field for each group of rows

flatten returns an array that flattens multiple arrays into one

```
SELECT user_id,
  collect_set(event_name) AS event_history,
  array_distinct(flatten(collect_set(items.item_id))) AS cart_history
FROM events
GROUP BY user_id
```



user_id	event_name	items
UA000000106494077	add_item	[{"coupon": null, "item_id": "M_PREM_Q", "item_name": "Premium Queen Mattress", "item_revenue_in_usd": 1795, "price_in_usd": 1795, "quantity": 1}, {"coupon": null, "item_id": "M_STAN_Q", "item_name": "Standard Queen Mattress", "item_revenue_in_usd": 1045, "price_in_usd": 1045, "quantity": 1}]
UA000000106494077	delivery	[]
UA000000106494077	email_coupon	[{"coupon": null, "item_id": "M_PREM_Q", "item_name": "Premium Queen Mattress", "item_revenue_in_usd": 1795, "price_in_usd": 1795, "quantity": 1}, {"coupon": null, "item_id": "M_STAN_Q", "item_name": "Standard Queen Mattress", "item_revenue_in_usd": 1045, "price_in_usd": 1045, "quantity": 1}, {"coupon": null, "item_id": "M_STAN_T", "item_name": "Standard Twin Mattress", "item_revenue_in_usd": 595, "price_in_usd": 595, "quantity": 1}]
UA000000106494077	main	[]
UA000000106494077	original	[]
UA000000106494077	premium	[{"coupon": null, "item_id": "M_PREM_Q", "item_name": "Premium Queen Mattress", "item_revenue_in_usd": 1795, "price_in_usd": 1795, "quantity": 1}, {"coupon": null, "item_id": "M_STAN_Q", "item_name": "Standard Queen Mattress", "item_revenue_in_usd": 1045, "price_in_usd": 1045, "quantity": 1}, {"coupon": null, "item_id": "M_STAN_T", "item_name": "Standard Twin Mattress", "item_revenue_in_usd": 595, "price_in_usd": 595, "quantity": 1}]
UA000000106494077	reviews	[]



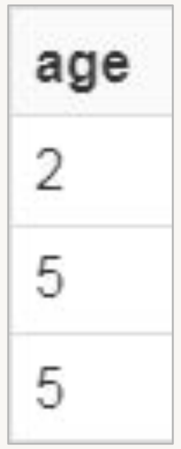
	user_id	event_history	cart_history
1	UA000000106494077	["add_item", "email_coupon", "main", "reviews", "original", "delivery", "premium"]	["M_PREM_Q", "M_STAN_Q", "M_STAN_T"]

Collection example

`collect_set` returns an array with duplicate elements eliminated

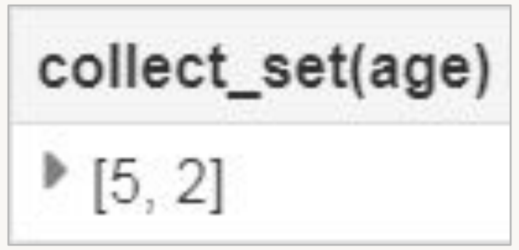
`collect_list` returns an array with duplicate elements intact

```
df
```



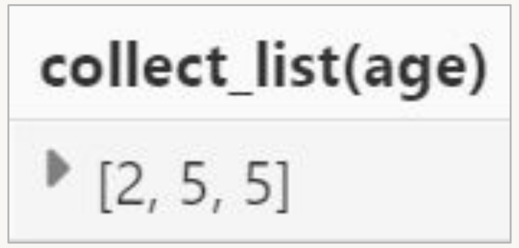
age
2
5
5

```
df.agg(collect_set('age'))
```



collect_set(age)
▸ [5, 2]

```
df.agg(collect_list('age'))
```



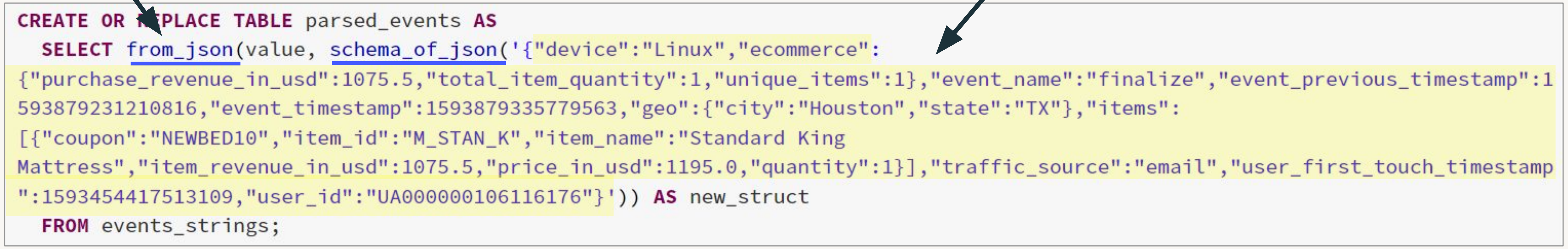
collect_list(age)
▸ [2, 5, 5]

Parse JSON strings into structs

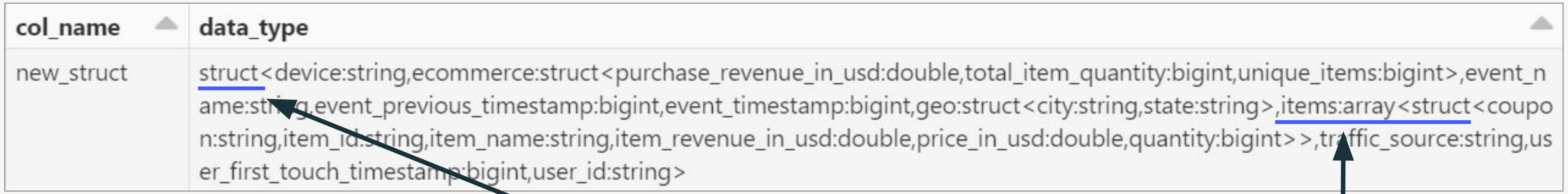
Create the schema to parse the JSON strings by providing an example JSON string from a row that has no nulls

from_json uses JSON schema returned by schema_of_json to convert a column of JSON strings into structs

This highlighted JSON string is taken from the value field of a single row of data



```
CREATE OR REPLACE TABLE parsed_events AS
  SELECT from_json(value, schema_of_json('{"device":"Linux","ecommerce":
{"purchase_revenue_in_usd":1075.5,"total_item_quantity":1,"unique_items":1},"event_name":"finalize","event_previous_timestamp":1
593879231210816,"event_timestamp":1593879335779563,"geo":{"city":"Houston","state":"TX"},"items":
[{"coupon":"NEWBED10","item_id":"M_STAN_K","item_name":"Standard King
Mattress","item_revenue_in_usd":1075.5,"price_in_usd":1195.0,"quantity":1}],"traffic_source":"email","user_first_touch_timestamp
":1593454417513109,"user_id":"UA000000106116176"}')) AS new_struct
  FROM events_strings;
```



col_name	data_type
new_struct	struct<device:string,ecommerce:struct<purchase_revenue_in_usd:double,total_item_quantity:bigint,unique_items:bigint>,event_name:string,event_previous_timestamp:bigint,event_timestamp:bigint,geo:struct<city:string,state:string>,items:array<struct<coupon:string,item_id:string,item_name:string,item_revenue_in_usd:double,price_in_usd:double,quantity:bigint>>,traffic_source:string,user_first_touch_timestamp:bigint,user_id:string>

Returns STRUCT column containing ARRAY of nested STRUCT

DE 2.5L: Reshape Data Lab (Optional)

DE 2.7A: SQL UDFs and Control Flow (Optional)

DE 2.7B: Python UDFs (Optional)

databricks