① Describe DATABASE EXTENDED db_name ;
This will show db-name, location where it is stored and its properties.

② Describe detail Table_Name also prints table details

③ Create external table:-

```
Create table table_name
using CSV options (
path 'path of .csv file'
header "true"
mode "FAILFAST"
);
```

④ 3 types of view:- 1) Regular 2) Temporary 3) Global Temporary

For temporary view specify TEMPORARY keyword after create

eg Create TEMPORARY View ViewName as
select * from tablename

For global temporary view: specify GLOBAL TEMPORARY keyword

⑤ Temporary views are not accessible in other sessions but global temp views are accessible in other sessions

But when we restart the cluster we won't be able to see GLOBAL TEMP view also

Whenever we want to query the global temp view, global_temp keyword should be added before global_temp view name
select * from global_temp.global temp viewname

⑥ CTE syntax

```
With cte_name (
new_colname1,
new_colname2,
new_colname3
) AS (
select
col1, col2, col3
from tbl_name ),
```

LTIMindtree

⑦ When u want to query a ~~file~~ data that is stored in parquet.

- select * from parquet.'location'

⑧ Describe TABLE Extended table_name ;-
to view table metadata

or

Describe detail table_name

- The temp view is not associated with any database. The temp view is only accessible in the current spark session
- Global temp view always register to global temp database

⑨ To query a single file

Select * from file_format.'/path/to/file'

⑩ Create a table which takes external data from .csv file

Create table tbl_name (col_1 datatype, col_2 datatype)
using CSV
options (header = "True", delimiter = "|")
Location "path to .csv file"

(11) We can manually refresh the cache of our data by running the REFRESH TABLE command

LTIMindtree

(12) Extracting data from SQL databases :-

```
= Create table table_name
  using org.apache.spark.sql.jdbc
  options (
    url "......",
    dbtable "{dbname}.table",
    user "{jdbcusername}",
    password "{jdbcpassword}"
  )
```

(13) CTAS statements are useful for external data ingestion from sources with WELL DEFINED SCHEMAS, such as Parquet files and tables.

So mostly we can't ingest .csv files with CTAS as there are limitations as we can't specify options parameters for .csv file

↓

So for .csv file :

First create temp view and write options, path parameters there and then use CTAS i.e create table tbl_name as select * from temp_viewname

CTAS statement do not support schema declaration.

(14) Databricks supports 2 types of constraints :

1) NOT NULL
2) CHECK

(12) Altering a table and adding CHECK constraint

ALTER TABLE table_name ADD CONSTRAINT constraint_name CHECK (date > '2020-01-01')

- 2 addition functions can be used in select

current_timestamp() 2 metadata
~~int file~~ input_file_name 3 function

(13) Cloning Delta lake tables

1) Deep clone fully copies data and metadata from a source into a target

- Create or replace table table_name DEEP clone tbl_name

2) SHALLOW CLONE just moves the transaction logs but the data doesn't move

(14) ⇒ Complete Overwrites

1) CRAS (Create or replace table)
statements fully replaces the contents of a table each time they execute

2) Insert Overwrite
Insert Overwrite table_name
select * from parquet.'pathname'

Insert Overwrite will fail if we try to change our schema

(15) Appending is much more efficient than overwriting

- Append rows

Insert into tbl_name
select * from parquet.'path'
This will append new rows result in duplicate records

(16) → MERGE UPDATES

Merge into target a
Using source b
ON {merge_condition}
When Matched then {matched-action}
When Not Matched then {not-matched-action}

LTIMindtree

(17) → LOAD INCREMENTALLY

Copy Into table_name
From "path"
FileFormat = PARQUET

for this u can add options: copy-parameter

copy into my_table
from '/path/to/files'
FILE_FORMAT = <format>
format_options ('inferSchema' = 'true')
copy_option ('mergeSchema' = 'true')

For this operation the data schema should be consistent

evolution

copy into support schema evolution

(18) → Cleaning Data

- count(col) does not count NULL values
but count(*) counts null values rows that contains for all columns

- To count null values, use the count_if function or where clause to filter for records where the value is NULL.

eg select
count_if(user_id is NULL) as missing_id
from tbl

(19) : syntax to traverse nested data structures

- Select value : device, value : geo : city from event_strings

(20) UDF

Create or replace function func_name (i_p name inp datatype)
RETURNS o/p datatype
RETURN ....

(21) EXISTS is a high order function:

Syntax :-

EXISTS (items, i -> i.item_name LIKE "%Mattress") AS Mattress

↑
It returns true or false

(22) checkpoint directory - the location for storing metadata about the stream. Checkpoints keep track of streaming progress, while the schema location tracks updates to the fields in the source dataset.

Readstream consist of formatting of data it includes

- format ("cloudfiles")
- option ("cloudFiles.format", "arg")
- option ("cloudFiles.schemaLocation", arg)
- load (data source)

Write stream includs
Checkpoint location, opt Merge schema,
table name.

r: ;f( )

r: (.

prows

q: ; ree

.src

(23) → windowing and watermarking can be used to add additional functionality to incremental workloads.

- Sorting is not supported on streaming data frames

(24) When writing to Delta Lake tables, we typically will only need to worry about 3 setting (Means these options needs to be included in writestream of autoloader)

1) Checkpointing 2) Output modes 3) Trigger intervals

- 1) Append (default) .outputMode("append")
- 2) Complete .outputMode("complete")
- 3) Unspecified .outputmode("append")
- b) Fixed interval micro-batches
- c) ~~Triggered micro-batch~~
- d) Triggered micro-batches

(25) cloudfiles
.schemaHints is used to specify the data type of column.

(26) DLT tables and views will always be preceded by LIVE keyword.

(27) The CONSTRAINT keyword introduces quality control.

ON Violations:
1) FAIL Update
2) DROP Row
3) Omitted

} These can be applied to streaming or DLT only

Trigger Type	Example	by default processing time = 500 ms
1) Unspecified		
2) Fixed interval micro batches	.trigger(processingTime = "2 minutes")	
3) Triggered micro-batch	.trigger(once = true)	
4) Triggered micro-batches	.trigger(availableNow = true)	

→ Selecting from version
select * from table_name VERSION AS OF 8

→ How to restore from version

LTIMindtree

Restore Table table_name to VERSION AS OF 8

→ set spark.databricks.delta.retentionDuration.check.enabled = false
if it set to false
Means without checking the duration of transaction file we can delete it directly

set spark.databricks.delta.vacuum.logging
-enabled = true

Means if it set to true the transaction log of vacuum we will be able to see in the describe history command.

→ Create a database with location specified

Create database if not exists db_name
location ';

→ If u specify the location of a TABLE while creating it then it becomes external (unmanaged) table

Create or Replace table table_name
LOCATION ' ' AS
select * from temp_delays

→ To read from data stored in parquet format

Select * from parquet.'location'

→ Managed VS external table

Create or replace table table_name AS select * from .... (Managed)	Create or replace table tbl_name LOCATION '....' AS select * from ..... (External)

→ Query a single file
Select * from file_format. 'path_to/file'
eg select * from json. 'dataset/raw/events/001.json'

→ Specifying the options for .csv

Create table tbl_name
(col_name1 col_name1 datatype, col_name2
col_name2 datatype)
USING CSV
OPTIONS (
header = "true",
delimiter = "|"
)
LOCATION "path to csv file"

→ Adding a table constraint: CHECK
NOT NULL

ALTER TABLE tbl_name ADD CONSTRAINT constraint_name
CHECK (columnname > '2020-01-01');

→ Deep Clone
Create or Replace table tbl_name
Deep Clone tbl_name.

→ For Shallow Clone replace Deep Clone with Shallow Clone.

→ Insert overwrite - To completely overwrite
Insert overwrite sales
Select * from parquet. `location`

→ To insert data into delta table from external table

Insert into deltatblname
select * from external_tblname

→ Interact with JSON data

select value:device, value:geo:city FROM events-strings.

schema_of_json

from_json - Used to convert string js. to json object

from_json

schema_of_json - To convert each value into proper Struct type. So that we can flatten the proper result in table.

Select from_json(value, schema_of_json('{obj:value}')) As json

Now to flatten

Select json.* from parsed_events;

Note: Whenever u want to convert a json file to table which has nested attributes then first convert a .json file to struct data type using from_json and then convert it into table

→ Directly reading from a .json file
- select value:device, value:geo:city from events

→ After converting or flattening the json file (converting to a struct type)

select geo.city from events.

Hint: [Before flattening use : colon to read and after flattening use . (dot) to read.

(*)5 DE 4.9 video

→ PIVOT: IN SQL [VERY VERY IMPORTANT] 2 D first

(converts rows to columns).

→ AUTO LOADER is used to incrementally load the data from cloud object storage to Delta table.

col

→ Syntax to read the data from a delta lake table and write it to streaming temp view

eg (spark.readstream
.table ("bronze")
.createOrReplaceTempView ("streaming_temp_view"))

→ Writing aggregated data to a delta table

~~create~~ spark.table ("customer_count_by_state_temp")

this view contains aggregated data

.writestream
.format ("delta")
.option ("checkpointLocation" location/path)
.outputMode ("complete")
.table ("customer_count_by_state")

→ If u are directly reading from a table or view then write it as spark.table(tbl_name) and if u are reading from some cloud based storage then write spark.readstream( )

→ Select *, current_timestamp() receipt_time, input_file_name() source_file from recordings_raw_temp

→ It will capture source file name

It will capture timestamp

→ Mostly the output mode of gold delta table is complete whereas the output mode of silver is append.

→ Delta Live table

To create a delta live table pipeline

Jobs → Delta live table → create a pipeline

In pipeline configurations

target - type the name of target database

Storage location - It is the location to store logs, tables and other information related to pipline execution

→ Delta live tables and views will always be preceded by the LIVE keyword

→ Incremental processing via Auto Loader (which uses the same processing model as structured Streaming), requires the addition of STREAMING keyword in the declaration as seen below. The cloud_files( ) method enables Auto Loader to be natively used with SQL. It takes the following parameters :-

1) source location
2) source data format
3) An arbitrarily sized array of optional reader options. In this case we set cloudfiles.inferColumnTypes to true

Syntax:- Create OR Refresh Streaming live table sales_order_raw

Comment "The raw sales orders, ingested from databricks datasets"

AS Select * from cloud_files ("source location path", "json", map("cloudfiles.inferColumnTypes", "true"))

→ 3 modes of constraint on violation

ON Violation	Behaviour
1) FAIL UPDATE	Pipeline failure when constraint is violated
2) DROP ROW	Discard records that violate constraints
3) Omitted/warn	Records violating constraints will be included (but violations will be reported in metrics)

→ References to DLT Tables and Views
References to other DLT tables and views will always include the live. prefix

→ References to Streaming Tables
References to streaming DLT tables use the STREAM( ), supplying the table name as an argument

→ Privileges in Data Explorer
1) All Privileges
2) Select
3) Modify
4) Read_Metadata
5) Usage
6) Create

→ The transfer of ownership of data objects like tables and databases can be carried by the current owner of the object OR Databricks admin only.

Syntax ALTER SCHEMA schema-name OWNER TO `user-name`

↑

Also Note, Schema and DATABASE can be used interchangeably.

→ Delta lake supports generated columns which are special type columns whose values are automatically generated based on a user-specified function over other columns in Delta table.

eg suppose from our dertime which has the timestamp datatype, I want to generate order date column which takes date directly from order time column

eg Create table orders (orderid INT, orderTime timestamp, orderdate DATE GENERATED ALWAYS AS (cast(orderTime as DATE))

Data object	Privileges
1) Catalog	Create, Usage
2) Schema	Create, Usage
3) ~~Select~~ Table	Select, Modify (insert, update, alter metadata)
4) View	Select
5) Function	Execute
6) Storage credential/ external location	Create table, Read files, Write files

→ If a select privilege on View is given then u don't need access to underlying source tables in views to just read data from a view.

→ In order to create a catalog, you should have metastore admin capability.