

Case Study

Name: Priyeshwar

Mail: priyesh2664@gmail.com

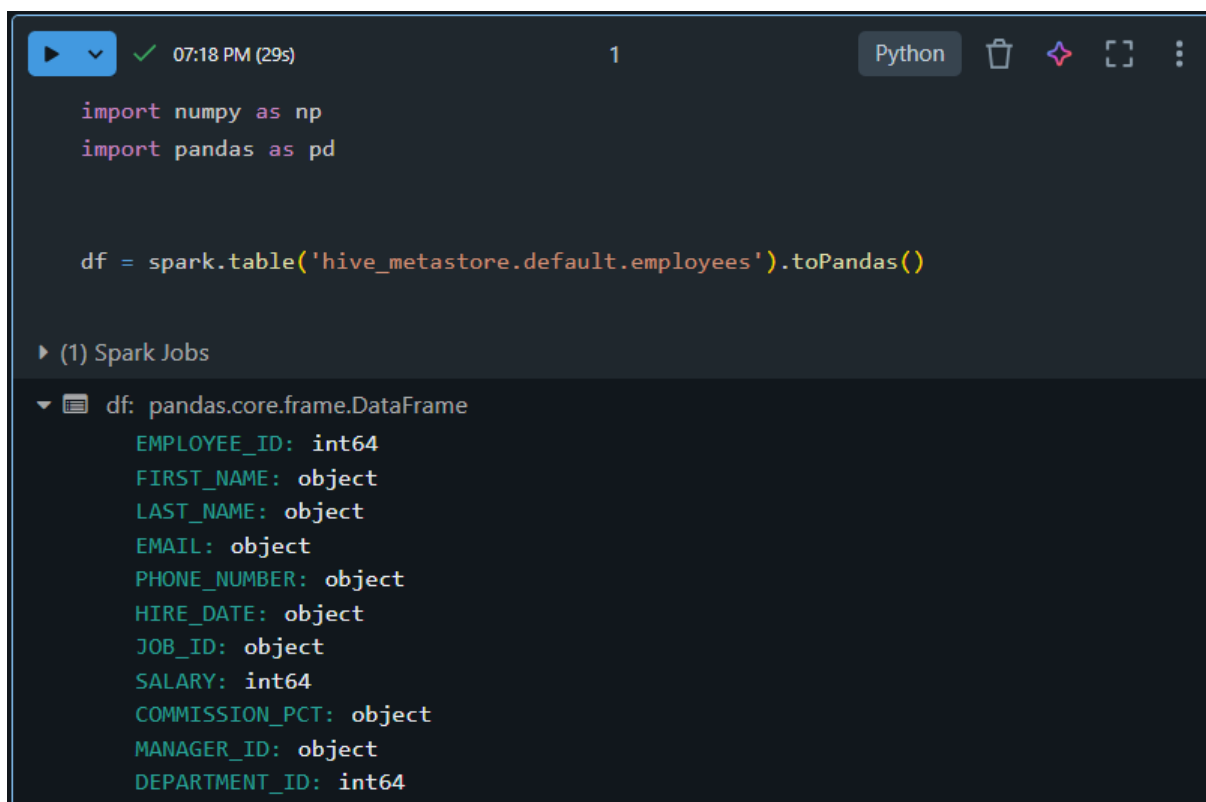
Load CSV file

Create a pandas DataFrame from the dataset for easier processing and visualization. Replace the file path below with your actual file location.

import numpy as np

import pandas as pd

df = spark.table('hive_metastore.default.employees').toPandas()



The screenshot shows a Jupyter Notebook interface. At the top, there's a toolbar with a play button, a checkmark, a timer showing '07:18 PM (29s)', a cell number '1', and a language dropdown set to 'Python'. Below the toolbar, the code cell contains the following Python code:

```
import numpy as np
import pandas as pd

df = spark.table('hive_metastore.default.employees').toPandas()
```

Below the code cell, there's a section titled '(1) Spark Jobs'. Underneath, a variable 'df' is shown with its type 'pandas.core.frame.DataFrame'. The columns and their data types are listed:

- EMPLOYEE_ID: int64
- FIRST_NAME: object
- LAST_NAME: object
- EMAIL: object
- PHONE_NUMBER: object
- HIRE_DATE: object
- JOB_ID: object
- SALARY: int64
- COMMISSION_PCT: object
- MANAGER_ID: object
- DEPARTMENT_ID: int64

Use pandas for data insights

Check shape of the DataFrame

df.shape

The df.shape command returns the number of rows and columns.

```
▶ ✓ 07:19 PM (<1s)
df.shape
(50, 11)
```

Generate descriptive statistics

df.describe()

The `df.describe()` command provides numerical summaries such as mean, standard deviation, min, max, and percentiles.

```
▶ ✓ 07:19 PM (<1s) 3
df.describe()
```

	EMPLOYEE_ID	SALARY	DEPARTMENT_ID
count	50.000000	50.000000	50.000000
mean	134.760000	6182.320000	57.600000
std	33.631594	4586.181772	25.11687
min	100.000000	2100.000000	10.000000
25%	112.250000	2725.000000	50.000000
50%	124.500000	4600.000000	50.000000
75%	136.750000	8150.000000	60.000000
max	206.000000	24000.000000	110.000000

Generate a data profile

display(df)

This displays a full interactive table of the data for profiling.

07:19 PM (1s) 4 Python

```
display(df)
```

Table + 🔍 🗑️ 📄

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PH
2	199	Douglas	Grant	DGRANT	650.50
3	200	Jennifer	Whalen	JWHALEN	515.12
4	201	Michael	Hartstein	MHARTSTE	515.12
5	202	Pat	Fay	PFAY	603.12
6	203	Susan	Mavris	SMAVRIS	515.12
7	204	Hermann	Baer	HBAER	515.12
8	205	Shelley	Higgins	SHIGGINS	515.12
9	206	William	Gietz	WGIEZT	515.12
10	100	Steven	King	SKING	515.12
11	101	Neena	Kochhar	NKOCHHAR	515.12
12	102	Lex	De Haan	LDEHAAN	515.12
13	103	Alexander	Hunold	AHUNOLD	590.42
14	104	Bruce	Ernst	BERNST	590.42

Clean the data

Remove duplicate data

```
duplicate_rows = df.duplicated().sum()
duplicate_columns = df.columns[df.columns.duplicated()].tolist()
print("Duplicate rows count:", duplicate_rows)
print("Duplicate columns:", duplicate_columns)
```

07:19 PM (<1s) 5

```
duplicate_rows = df.duplicated().sum()
duplicate_columns = df.columns[df.columns.duplicated()].tolist()
print("Duplicate rows count:", duplicate_rows)
print("Duplicate columns:", duplicate_columns)
```

Duplicate rows count: 0
Duplicate columns: []

Handle null or missing values

```
df = df.replace('-', np.nan)
```

```
df = df.fillna(0)
```

```
07:19 PM (<1s) 6 Python
df = df.replace('-', np.nan)
df = df.fillna(0)

df: pandas.core.frame.DataFrame = [EMPLOYEE_ID: int64, FIRST_NAME: object ... 9 more fields]
```

Reformat dates

```
df['HIRE_DATE'] = pd.to_datetime(df['HIRE_DATE'], format='%d-%b-%y', errors='coerce')
```

```
display(df)
```

```
07:19 PM (<1s) 7 Python
df['HIRE_DATE'] = pd.to_datetime(df['HIRE_DATE'], format='%d-%b-%y',
errors='coerce')
display(df)
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PH
1	198	Donald	OConnell	DOCONNEL	650.50
2	199	Douglas	Grant	DGRANT	650.50
3	200	Jennifer	Whalen	JWHALEN	515.12
4	201	Michael	Hartstein	MHARTSTE	515.12
5	202	Pat	Fay	PFAY	603.12
6	203	Susan	Mavris	SMAVRIS	515.12
7	204	Hermann	Baer	HBAER	515.12
8	205	Shelley	Higgins	SHIGGINS	515.12
9	206	William	Gietz	WGIETZ	515.12
10	100	Steven	King	SKING	515.12

Filter for specific conditions

Employees with salary > 5000

```
high_salary = df[df['SALARY'] > 5000]
```

```
# Employees in department 20
```

```
dept_20 = df[df['DEPARTMENT_ID'] == 20]
```



The screenshot shows a Jupyter Notebook interface. At the top, there's a code cell with the following Python code:

```
high_salary = df[df['SALARY'] > 5000]
display(high_salary)
dept_20 = df[df['DEPARTMENT_ID'] == 20]
```

 Below the code cell, there are two output messages:

```
dept_20: pandas.core.frame.DataFrame = [EMPLOYEE_ID: int64, FIRST_NAME: object ... 9 more fields]
```

 and

```
high_salary: pandas.core.frame.DataFrame = [EMPLOYEE_ID: int64, FIRST_NAME: object ... 9 more fields]
```

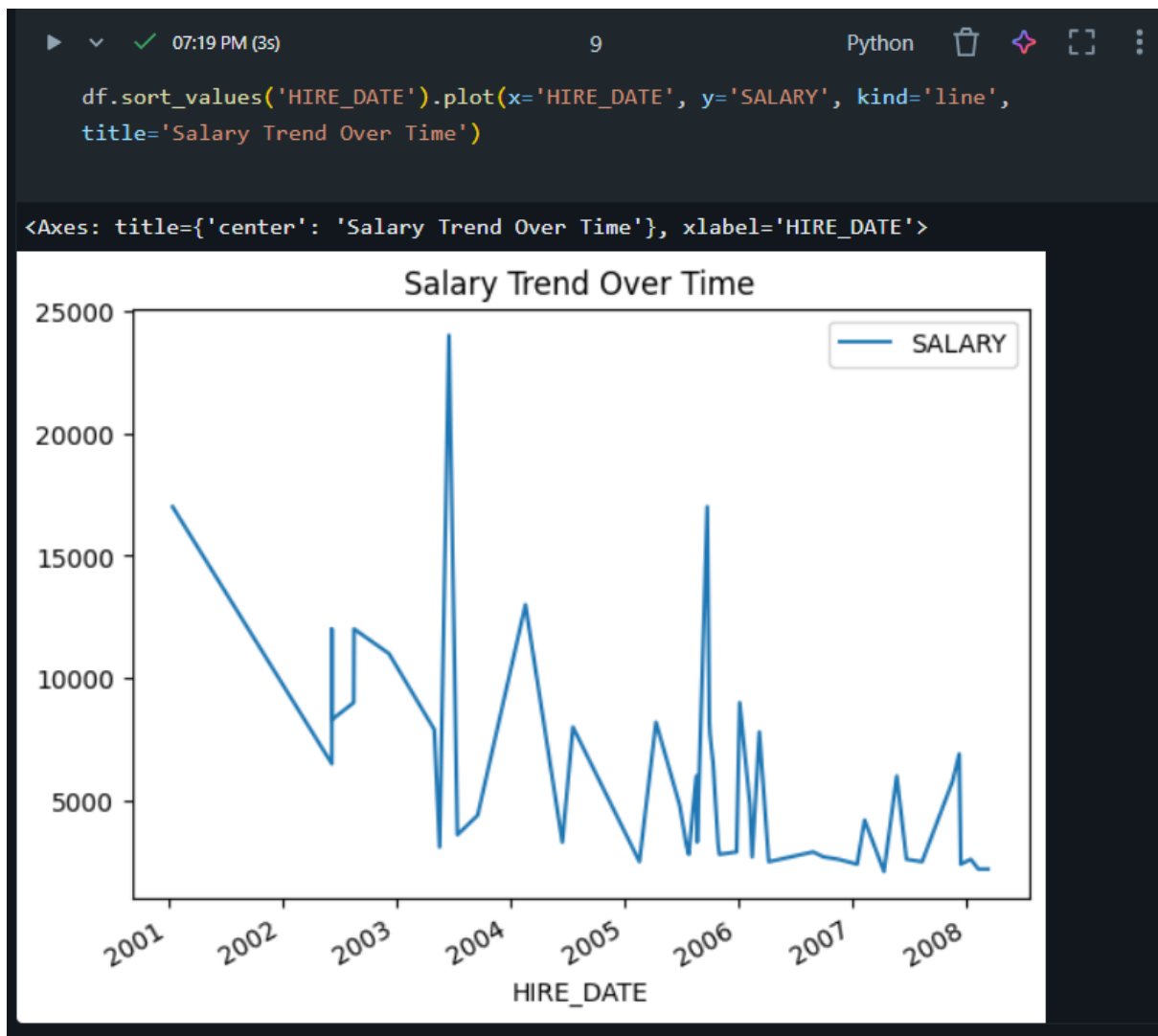
. Below these messages, there is a table view of the `dept_20` DataFrame. The table has 6 columns: `EMPLOYEE_ID`, `FIRST_NAME`, `LAST_NAME`, `EMAIL`, and `PH`. The first 5 rows of the table are displayed, showing employees with IDs 201 through 205.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PH
1	201	Michael	Hartstein	MHARTSTE	515.12
2	202	Pat	Fay	PFAY	603.12
3	203	Susan	Mavris	SMAVRIS	515.12
4	204	Hermann	Baer	HBAER	515.12
5	205	Shelley	Higgins	SHIGGINS	515.12

Create visualizations using the dataset

Line Chart

```
df.sort_values('HIRE_DATE').plot(x='HIRE_DATE', y='SALARY', kind='line', title='Salary Trend Over Time')
```



Area Chart

```
df.groupby('DEPARTMENT_ID')['SALARY'].sum().plot(kind='area', title='Total Salary by Department')
```

05:55 PM (<1s)

10

```
df.groupby('DEPARTMENT_ID')['SALARY'].sum().plot(kind='area', title='Total Salary  
by Department')
```

<Axes: title={'center': 'Total Salary by Department'}, xlabel='DEPARTMENT_ID'>



Delta Table Operations

Create a Delta Table

Convert pandas DataFrame to Spark DataFrame

```
spark_df = spark.createDataFrame(df)
```

Write as Delta table

```
spark_df.write.format("delta").mode("overwrite").saveAsTable("hive_metastore.default.employee_d  
elta")
```

```
▶ 07:20 PM (8s) 11 Python
# Convert pandas DataFrame to Spark DataFrame
spark_df = spark.createDataFrame(df)

# Write as Delta table
spark_df.write.format("delta").mode("overwrite").saveAsTable("hive_metastore.
default.employee_delta")

▶ (1) Spark Jobs
▶ spark_df: pyspark.sql.dataframe.DataFrame = [EMPLOYEE_ID: long, FIRST_NAME: string ... 9 more
fields]
```

Merge & Upsert to a Delta Table

```
from delta.tables import DeltaTable
```

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType,
DateType
```

```
from datetime import datetime
```

```
# Define schema explicitly
```

```
schema = StructType([
    StructField("EMPLOYEE_ID", IntegerType(), False),
    StructField("FIRST_NAME", StringType(), True),
    StructField("LAST_NAME", StringType(), True),
    StructField("EMAIL", StringType(), True),
    StructField("PHONE_NUMBER", StringType(), True),
    StructField("HIRE_DATE", DateType(), True),
    StructField("JOB_ID", StringType(), True),
    StructField("SALARY", DoubleType(), True),
    StructField("COMMISSION_PCT", DoubleType(), True),
    StructField("MANAGER_ID", IntegerType(), True),
    StructField("DEPARTMENT_ID", IntegerType(), True)
])
```

```
# Create source DataFrame with explicit schema
```



```
source = spark.createDataFrame([
    (205, "Alex", "Smith", "ASMITH", "650.123.9999", datetime.strptime("12-Jan-09", "%d-%b-%y"),
    "IT_PROG", 9000.0, None, 101, 60)
], schema=schema)
```

```
# Merge into Delta table
```

```
target = DeltaTable.forName(spark, "hive_metastore.default.employee_delta")
```

```
(
    target.alias("t")
    .merge(source.alias("s"), "t.EMPLOYEE_ID = s.EMPLOYEE_ID")
    .whenMatchedUpdateAll()
    .whenNotMatchedInsertAll()
    .execute()
)
```

```

from delta.tables import DeltaTable
from pyspark.sql.types import StructType, StructField, IntegerType, StringType,
DoubleType, DateType
from datetime import datetime

# Define schema explicitly
schema = StructType([
    StructField("EMPLOYEE_ID", IntegerType(), False),
    StructField("FIRST_NAME", StringType(), True),
    StructField("LAST_NAME", StringType(), True),
    StructField("EMAIL", StringType(), True),
    StructField("PHONE_NUMBER", StringType(), True),
    StructField("HIRE_DATE", DateType(), True),
    StructField("JOB_ID", StringType(), True),
    StructField("SALARY", DoubleType(), True),
    StructField("COMMISSION_PCT", DoubleType(), True),
    StructField("MANAGER_ID", IntegerType(), True),
    StructField("DEPARTMENT_ID", IntegerType(), True)
])

# Create source DataFrame with explicit schema
source = spark.createDataFrame([
    (205, "Alex", "Smith", "ASMITH", "650.123.9999", datetime.strptime(
        "12-Jan-09", "%d-%b-%y"), "IT_PROG", 9000.0, None, 101, 60)
], schema=schema)

```


```

# Merge into Delta table
target = DeltaTable.forName(spark, "hive_metastore.default.employee_delta")

(
    target.alias("t")
    .merge(source.alias("s"), "t.EMPLOYEE_ID = s.EMPLOYEE_ID")
    .whenMatchedUpdateAll()
    .whenNotMatchedInsertAll()
    .execute()
)

```

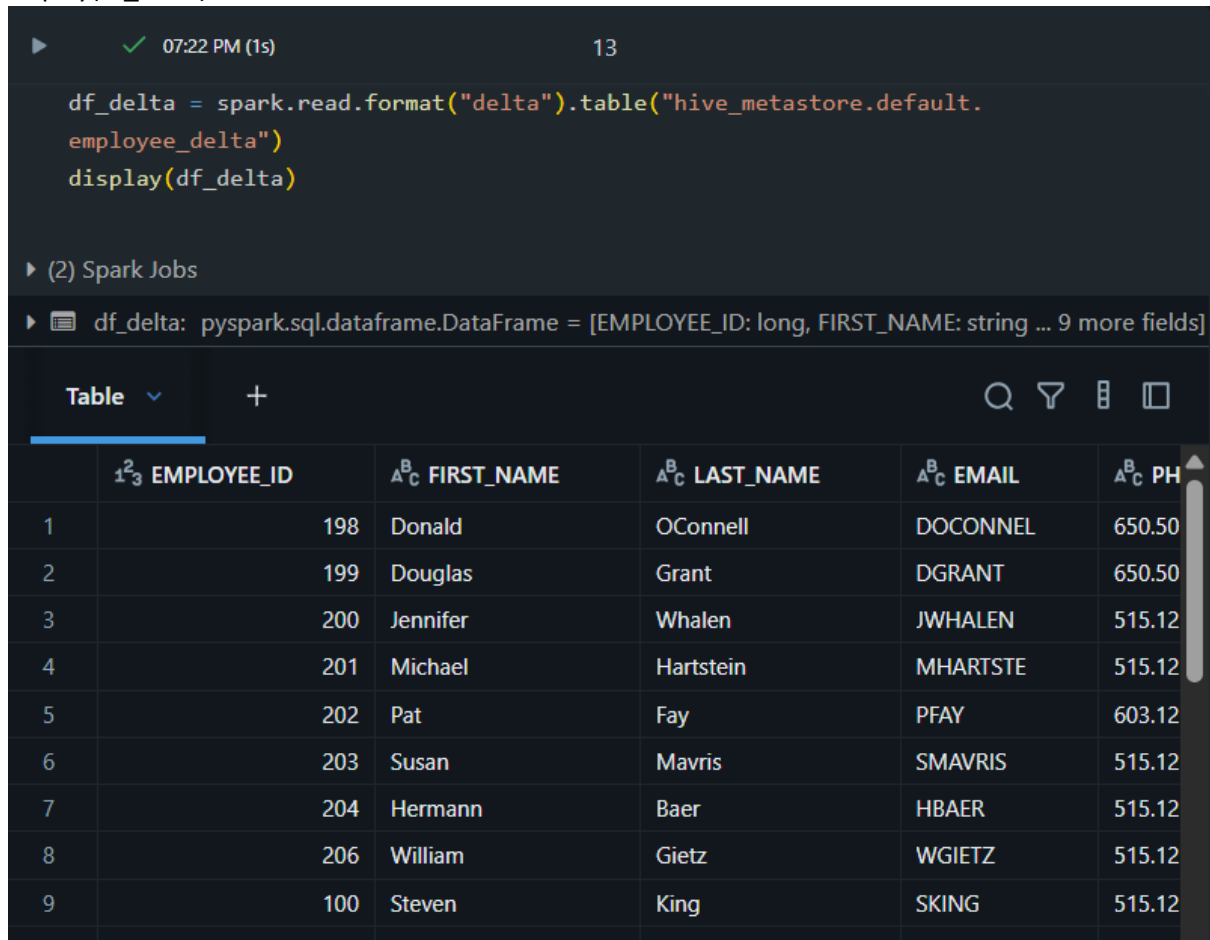
▶ (8) Spark Jobs

▶  source: pyspark.sql.dataframe.DataFrame = [EMPLOYEE_ID: integer, FIRST_NAME: string ... 9 more fields]

Read a Delta Table

```
df_delta = spark.read.format("delta").table("hive_metastore.default.employee_delta")
```

display(df_delta)



The screenshot shows a Jupyter Notebook interface. At the top, a status bar indicates a successful execution at 07:22 PM (1s) with cell number 13. The code cell contains the following Spark code:

```
df_delta = spark.read.format("delta").table("hive_metastore.default.employee_delta")
display(df_delta)
```

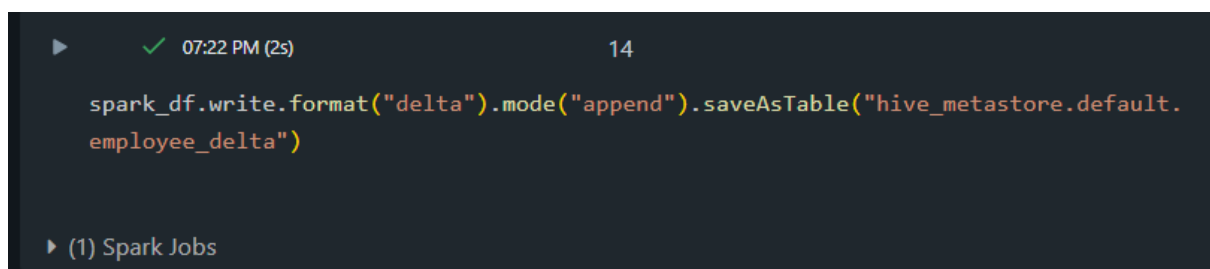
Below the code, a summary of the Spark job is shown: (2) Spark Jobs. A message indicates the DataFrame schema: df_delta: pyspark.sql.dataframe.DataFrame = [EMPLOYEE_ID: long, FIRST_NAME: string ... 9 more fields].

The table view is displayed below the summary. It has a header row with columns: EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, and PH. The table contains 10 rows of data, with the first 9 rows visible in the screenshot. The columns are labeled with small icons: a blue '1' for EMPLOYEE_ID, a blue '2' for FIRST_NAME, a blue '3' for LAST_NAME, a blue '4' for EMAIL, and a blue '5' for PH.

	1 EMPLOYEE_ID	2 FIRST_NAME	3 LAST_NAME	4 EMAIL	5 PH
1	198	Donald	OConnell	DOCONNEL	650.50
2	199	Douglas	Grant	DGRANT	650.50
3	200	Jennifer	Whalen	JWHALEN	515.12
4	201	Michael	Hartstein	MHARTSTE	515.12
5	202	Pat	Fay	PFAY	603.12
6	203	Susan	Mavris	SMAVRIS	515.12
7	204	Hermann	Baer	HBAER	515.12
8	206	William	Gietz	WGIETZ	515.12
9	100	Steven	King	SKING	515.12

Write to a Delta Table

```
spark_df.write.format("delta").mode("append").saveAsTable("hive_metastore.default.employee_delta")
```



The screenshot shows a Jupyter Notebook interface. At the top, a status bar indicates a successful execution at 07:22 PM (2s) with cell number 14. The code cell contains the following Spark code:

```
spark_df.write.format("delta").mode("append").saveAsTable("hive_metastore.default.employee_delta")
```

Below the code, a summary of the Spark job is shown: (1) Spark Jobs.

Update a Delta Table

```
target.update(
    condition="DEPARTMENT_ID = 50",
    set={"SALARY": "SALARY + 500"}
)
```

```
▶ 07:23 PM (4s) 15

target.update(
  condition="DEPARTMENT_ID = 50",
  set={"SALARY": "SALARY + 500"}
)

▶ (6) Spark Jobs
```

Display Table History

```
display(spark.sql("DESCRIBE HISTORY hive_metastore.default.employee_delta"))
```

```
▶ 07:23 PM (1s) 16

display(spark.sql("DESCRIBE HISTORY hive_metastore.default.employee_delta"))

▶ (1) Spark Jobs
```

	version	timestamp	userId	userName
1	5	2025-08-12T13:53:11.000+00:00...	141397643554594	azuser4022_mml.local@techa
2	4	2025-08-12T13:53:08.000+00:00...	141397643554594	azuser4022_mml.local@techa
3	3	2025-08-12T13:52:47.000+00:00...	141397643554594	azuser4022_mml.local@techa
4	2	2025-08-12T13:51:49.000+00:00...	141397643554594	azuser4022_mml.local@techa
5	1	2025-08-12T13:50:08.000+00:00...	141397643554594	azuser4022_mml.local@techa
6	0	2025-08-12T12:26:07.000+00:00...	141397643554594	azuser4022_mml.local@techa

6 rows | 1.39s runtime Refreshed 17 minutes ago

Query Earlier Version (Time Travel)

```
df_old = spark.read.format("delta").option("versionAsOf",
0).table("hive_metastore.default.employee_delta")

display(df_old)
```

07:23 PM (1s) 17 Python

```
df_old = spark.read.format("delta").option("versionAsOf", 0).table("hive_metastore.default.employee_delta")
display(df_old)
```

▶ (1) Spark Jobs

df_old: pyspark.sql.dataframe.DataFrame = [EMPLOYEE_ID: long, FIRST_NAME: string ... 9 more fields]

Table + 🔍 ⚙️ 📄

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PH
1	198	Donald	OConnell	DOCONNEL	650.50
2	199	Douglas	Grant	DGRANT	650.50
3	200	Jennifer	Whalen	JWHALEN	515.12
4	201	Michael	Hartstein	MHARTSTE	515.12
5	202	Pat	Fay	PFAY	603.12
6	203	Susan	Mavris	SMAVRIS	515.12
7	204	Hermann	Baer	HBAER	515.12
8	205	Shelley	Higgins	SHIGGINS	515.12
9	206	William	Gietz	WGIEZT	515.12
10	100	Steven	King	SKING	515.12
11	101	Neena	Kochhar	NKOCHHAR	515.12

Optimize a Delta Table

```
spark.sql("OPTIMIZE hive_metastore.default.employee_delta")
```

07:25 PM (2s) 18 Python

```
spark.sql("OPTIMIZE hive_metastore.default.employee_delta")
```

▶ (3) Spark Jobs

```
>,partitionsOptimized:bigint,zOrderStats:struct<strategyName:string,inputCubeFiles:struct<num:bigint,size:bigint>,inputOtherFiles:struct<num:bigint,size:bigint>,inputNumCubes:bigint,mergedFiles:struct<num:bigint,size:bigint>,numOutputCubes:bigint,mergedNumCubes:bigint>,clusteringStats:struct<inputZCubeFiles:struct<numFiles:bigint,size:bigint>,inputOtherFiles:struct<numFiles:bigint,size:bigint>,inputNumZCubes:bigint,mergedFiles:struct<numFiles:bigint,size:bigint>,numOutputZCubes:bigint>,numBins:bigint,numBatches:bigint,totalConsideredFiles:bigint,totalFilesSkipped:bigint,preserveInsertionOrder:boolean,numFilesSkippedToReduceWriteAmplification:bigint,numBytesSkippedToReduceWriteAmplification:bigint,startTimeMs:bigint,endTimeMs:bigint,totalClusterParallelism:bigint,totalScheduledTasks:bigint,autoCompactParallelismStats:struct<maxClusterActiveParallelism:bigint,minClusterActiveParallelism:bigint,maxSessionActiveParallelism:bigint,minSessionActiveParallelism:bigint>,deletionVectorStats:struct<numDeletionVectorsRemoved:bigint,numDeletionVectorRowsRemoved:bigint>,recompressionCodec:string,numTableColumns:bi
```

Clean Up Snapshots with VACUUM

```
spark.sql("VACUUM hive_metastore.default.employee_delta RETAIN 168 HOURS")
```



The screenshot shows a Databricks console interface. At the top, there is a green checkmark icon, a timestamp '07:26 PM (51s)', and a value '20'. Below this, the command `spark.sql("VACUUM hive_metastore.default.employee_delta RETAIN 168 HOURS")` is displayed in a monospaced font. Underneath the command, there is a section labeled '(18) Spark Jobs'. At the bottom of the console, the return type `DataFrame[path: string]` is shown.