Ex No: 6 Implement Hive Databases, Tables, Views, Functions, and Indexes

## AIM:

To create and demonstrate operations on Hive databases including table creation, views, indexes, and user-defined functions (UDFs) using a simulated Hive-like environment in Python.

## Algorithm :

1. **Create Hive Database:**
   * Initialize a database (simulated here using SQLite for demonstration).

## Create Table:

* + Define a table to store sample weather data with columns for record\_id, year, and temperature\_c.

## Insert Data:

* + Load random weather data for multiple years into the table.

## Create Index:

* + Create an index on the year column to optimize queries.

## Create View:

* + Define a view (positive\_temps) showing only records with temperature greater than 0°C.

## Create User-Defined Function (UDF):

* + Implement a Python function to convert Celsius to Fahrenheit.
  + Register the UDF within the database.

## Query and Reporting:

* + Generate reports of **min/max temperatures per year** using SQL queries.
  + Use the UDF within queries to convert values dynamically.

## Python Implementation

import pandas as pd import sqlite3 import random

from contextlib import contextmanager

# ================================

# 2. Generate Sample Weather Data

# ================================

def generate\_sample\_data(num\_records=1000): years = list(range(1900, 2021))

data = {

'record\_id': range(1, num\_records + 1),

'year': [random.choice(years) for \_ in range(num\_records)], 'temperature\_c': [random.uniform(-50, 50) for \_ in range(num\_records)]

}

return pd.DataFrame(data)

# ================================

# 3. Simulate Hive Database & Table in SQLite # ================================

@contextmanager

def sqlite\_connection(db\_name): conn = sqlite3.connect(db\_name) try:

yield conn finally:

conn.close()

def setup\_hive\_like\_db(): db\_name = 'weather\_hive.db'

df = generate\_sample\_data(1000)

with sqlite\_connection(db\_name) as conn: # Create Hive-like Table

df.to\_sql('weather\_data', conn, if\_exists='replace', index=False)

# Create Index (simulating Hive CREATE INDEX) conn.execute('CREATE INDEX idx\_year ON weather\_data(year)')

# Create View (simulating Hive CREATE VIEW) conn.execute('''

CREATE VIEW positive\_temps AS SELECT record\_id, year, temperature\_c FROM weather\_data

WHERE temperature\_c > 0

''')

print(f"Database '{db\_name}', table 'weather\_data', index 'idx\_year', and view 'positive\_temps' created successfully.")

# ================================

# 4. Create Hive-Like UDF (Function)

# ================================

def celsius\_to\_fahrenheit(temp\_c): return (temp\_c \* 9/5) + 32

def register\_udf(conn):

conn.create\_function('c\_to\_f', 1, celsius\_to\_fahrenheit)

print("User Defined Function (UDF) 'c\_to\_f' registered successfully.")

# ================================

# 5. Generate Weather Report

# ================================

def generate\_weather\_report(): db\_name = 'weather\_hive.db'

with sqlite\_connection(db\_name) as conn: register\_udf(conn)

# Query Table: Min/Max per Year

query\_table = ''' SELECT year,

MIN(temperature\_c) AS min\_temp\_c, MAX(temperature\_c) AS max\_temp\_c

FROM weather\_data GROUP BY year ORDER BY year

'''

report\_df = pd.read\_sql\_query(query\_table, conn)

# Query View: Max Temp in Fahrenheit using UDF query\_view = '''

SELECT year,

c\_to\_f(MAX(temperature\_c)) AS max\_temp\_f FROM positive\_temps

GROUP BY year ORDER BY year

'''

view\_df = pd.read\_sql\_query(query\_view, conn)

# Merge Both Results

result = report\_df.merge(view\_df, on='year', how='left') result['max\_temp\_f'] = result['max\_temp\_f'].round(1) result['min\_temp\_c'] = result['min\_temp\_c'].round(1) result['max\_temp\_c'] = result['max\_temp\_c'].round(1)

return result

# ================================

# 6. Main Execution

# ================================

if name == " main ":

print("Setting up Hive-like environment...") setup\_hive\_like\_db()

print("\nGenerating Weather Temperature Statistics Report...") report = generate\_weather\_report()

print("\n=== Weather Report ===")

print("Year\tMin Temp (°C)\tMax Temp (°C)\tMax Temp (°F)") print("-" \* 50)

for \_, row in report.iterrows(): print(f"{int(row['year'])}\t{row['min\_temp\_c']}\t\t{row['max\_temp\_c']}\t\t{row['max\_temp\_f']}")

print("\nSample Data from View (positive\_temps):") with sqlite\_connection('weather\_hive.db') as conn:

sample\_view = pd.read\_sql\_query('SELECT \* FROM positive\_temps LIMIT 5', conn) print(sample\_view)

# Expected Output:

Setting up Hive-like environment...

Database 'weather\_hive.db', table 'weather\_data', index 'idx\_year', and view 'positive\_temps' created successfully.

Generating Weather Temperature Statistics Report...

User Defined Function (UDF) 'c\_to\_f' registered successfully.

=== Weather Report ===

Year Min Temp (°C) Max Temp (°C) Max Temp (°F)

|  |  |  |  |
| --- | --- | --- | --- |
| 1900 | -47.6 | 49.9 | 121.9 |
| 1901 | -49.1 | 48.7 | 119.7 |
| 1902 | -45.2 | 47.8 | 118.0 |

... (truncated) ...

Sample Data from View (positive\_temps):

record\_id year temperature\_c

0 2 1910 10.34

1 12 1954 24.76

2 25 1998 3.25

3 45 2009 47.92

4 52 1965 17.13

# Result:

The Hive Experiment was successfully created using Python and SQLite to demonstrate database creation, tables, views, indexes, and user-defined functions. It efficiently generated analytical reports showing yearly temperature statistics in both Celsius and Fahrenheit.