

## **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:
  - o Initialize a database (simulated here using SQLite for demonstration).
2. Create Table:
  - o Define a table to store sample weather data with columns for record\_id, year, and temperature\_c.
- 3 Insert Data:
  - o Load random weather data for multiple years into the table.
- . Create Index:
  - o Create an index on the year column to optimize queries.
- 4 Create View:
  - o Define a view (positive\_temps) showing only records with temperature greater than 0°C.
5. Create User-Defined Function (UDF):
  - o Implement a Python function to convert Celsius to Fahrenheit.
  - o Register the UDF within the database.
- .7. Query and Reporting:
  - o Generate reports of min/max temperatures per year using SQL queries.
  - o 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"\t{int(row['year'])}\t{row['min_temp_c']}\t{row['max_temp_c']}\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...

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

====Weather Report ===

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

190	-47.6	49.9	121.9
0	-49.1	48.7	119.7
190	-45.2	47.8	118.0

.1..(truncated) ...

\$0pleData from View (positive\_temps):

2

	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.