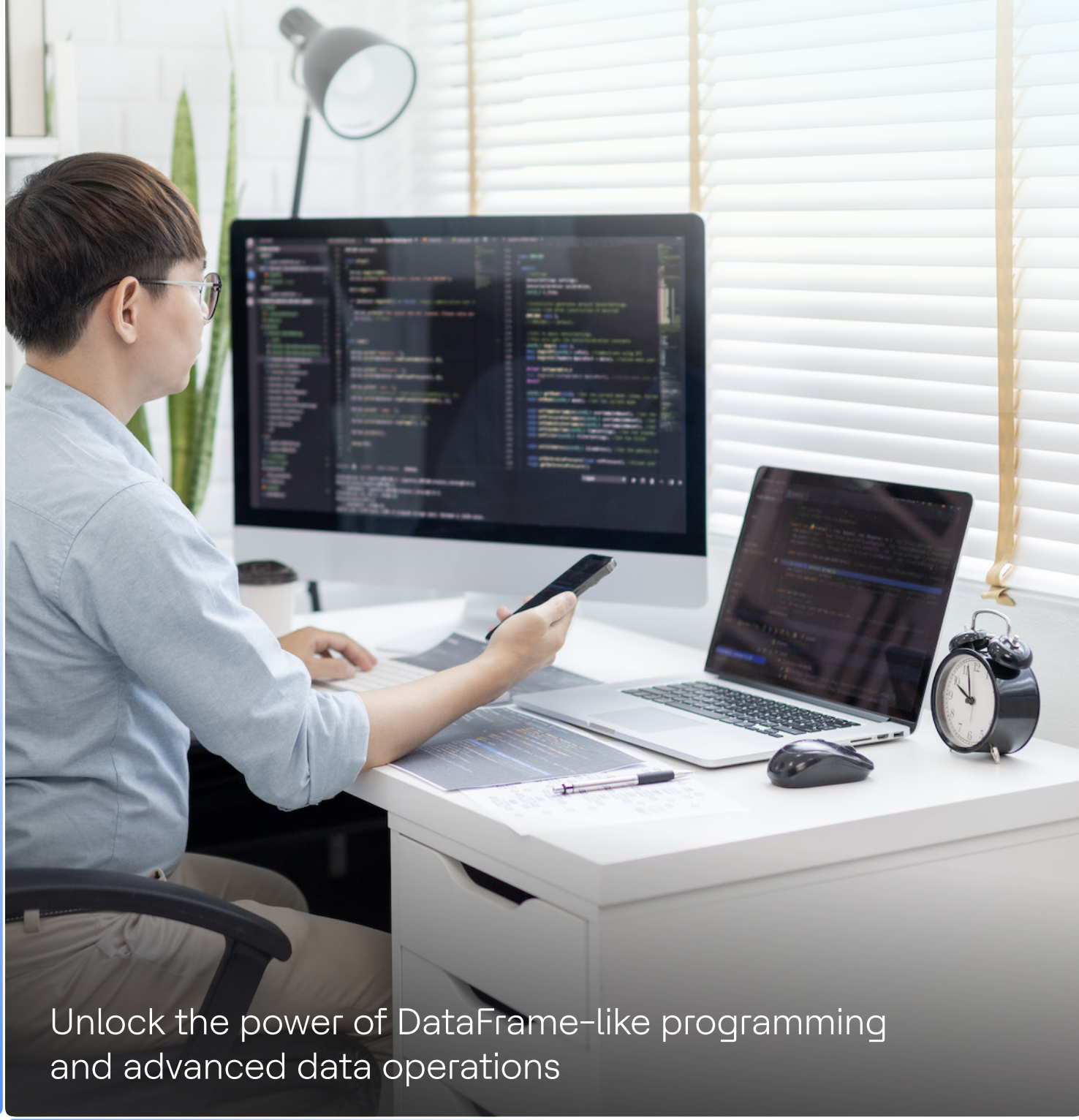
HCLTech | Supercharging Progress™

Accelerate data processing with Snowflake's Snowpark developer framework



Unlock the power of DataFrame-like programming and advanced data operations

Introduction

Snowflake's Snowpark is a developer framework that brings Data frame-like programming, using languages like Python, Scala and Java to perform data processing operations inside Snowflake compute.

The Snowpark library will help eliminate the need to move data to the system where the application code runs.

It allows for custom code to be defined as user-defined functions (UDFs) and/or stored procedures and use that code in Data frame operations.

Developers can use toolsets like version control, development IDE, debugging tools, own custom libraries and also manage package dependencies easily using Anaconda.

Snowflake's unique multi-cluster shared data architecture powers the performance, elasticity and governance of Snowpark.

Snowpark-optimized warehouse

Snowpark provides 16 times more memory per node and 10 times more storage for compute-intensive workloads. These are available in sizes starting from medium till 6X-Large across all regions in all public cloud (AWS, Azure, GCP).

This type of warehouse is recommended for large memory workloads such as ML training, stored procedures, UDF and UDTF.

Use cases

Data science and machine learning:

ML models can be built and executed on Snowflake compute, using Data frame API and server-side runtime capabilities.

Data preparation:

Snowpark can be used for data cleansing, standardization and feature engineering using SQL, Python, Scala or Java. For complex data transformations, reusable and readable code can be created using the functional programming paradigm.

Data-intensive applications:

Snowpark lets applications run on Snowflake compute. Combining Snowflake-native applications, secure data sharing and Snowpark both allow customers to process data in a secure and governed way.

Data visualization:

Using libraries as an example, matplotlib, among others, is available from Anaconda python distribution. Through this, users can create visualizations and analyze data.

Syntax differences in Spark and Snowpark

Syntactically, Snowpark is very close to Spark and Databricks coding practices. Below, however, are a few differences:

Read file from cloud storage (AWS S3)

Spark/Databricks	Snowpark
Create an Instance profile to establish connectivity from Databricks notebook to S3	Create a storage integration for AWS S3 to Snowflake connectivity
Create a mount point and read data into data frame based on file types	Create an external stage and read data into data frame based on file types
df = (spark.read.format("csv") .option("header", "true").option("inferSchema", "true").load("dbfs:/mnt/mntpoc/out/STG_GDP_DATA_OUT/data_geo.csv")	df_json = session.read.json("@my_stage2/data1.json") df_catalog = session.read .schema(StructType([StructField("name", StringType()), StructField("age", IntegerType())])) .csv("@stage/some_dir")

Write Data frame to table

Spark/Databricks

Write output to a file

```
(usersDF.write.option("compression", "snappy")
 .mode("overwrite").parquet(OutputPath))
```

Write output to a table

```
eventsDF.write.mode("overwrite").saveAsTable
("events_p")
```

Snowpark

Write output to a file

```
copy_result=df.write.copy_into_location
("@my_stage_s3/data", file_format_type=
"parquet", header=True, overwrite=True)
```

Write output to a table

```
Df.write.mode("overwrite").save_as_table
("table1")
```

Differences in transform functions

Spark/Databricks

- If()
- Boolean(col_nm)
- collect_list(col_name)
- concat_ws(", ", collect_list(col_name))
- date_sub(created_date, 3)
- from_unix_time(unixtime)
- datediff(updated_date, created_d
- newDF = df.select("user_id", col("geo.city")
  .alias("city"))

Snowpark

- Iff()
- Col_nm::Boolean
- array_agg(col_name)
- listagg(col_name, ', ')
- dateadd(day, -3, created_date)
- unixtime::timestamp_ntz
- datediff(day, created_date, updated_date)
- df_new = df.select("id", col("parent_id")
  .as_("pid"))

The HCLTech differentiator

Migrate to Snowflake leveraging HCL's ADvantage Migrate Suite.
ADvantage Migrate is a one-stop solution to modernize the entire data landscape within an enterprise.

Our 3-step automation process with pre-engineered products

1 Gateway Suite

Discover, Analyze and Auto convert the legacy code & DB schemas using HCL Gateway Suite

Gateway Suite:
Embed automation in modernization process

2 Sketch

Modernize the data pipelines and migrate data using HCL Sketch

Sketch:
Platform agnostic configuration driven data processing

3 Gatekeeper

Achieve zero-touch automated Data Reconciliation and Testing post migration using HCL Gatekeeper

Gatekeeper:
Auto reconcile, test and setup test driven development and continuous testing