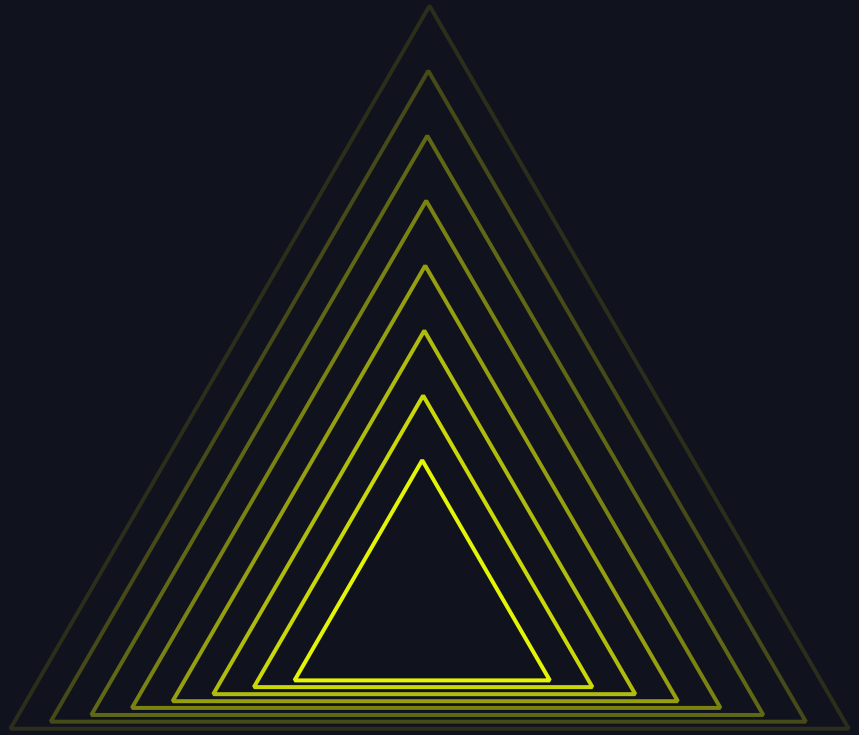
DATA+AI SUMMIT BY databricks

EXPLORING UDTFS IN PYSPARK

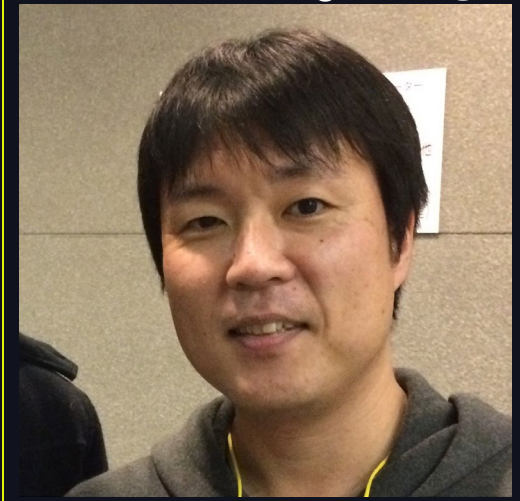


Takuya Ueshin, Haejoon Lee
Data+AI Summit 2024

Introductions

Takuya Ueshin

Sr. Software Engineer @ Databricks



Haejoon Lee

Software Engineer @ Databricks

Agenda

- What are UDTFs?
- Capabilities and areas for improvement in UDTFs
- Introduction to polymorphism
- Making UDTFs polymorphic
- Example/demo
- Conclusion

WHAT ARE UDTFs (User-Defined Table Functions)?

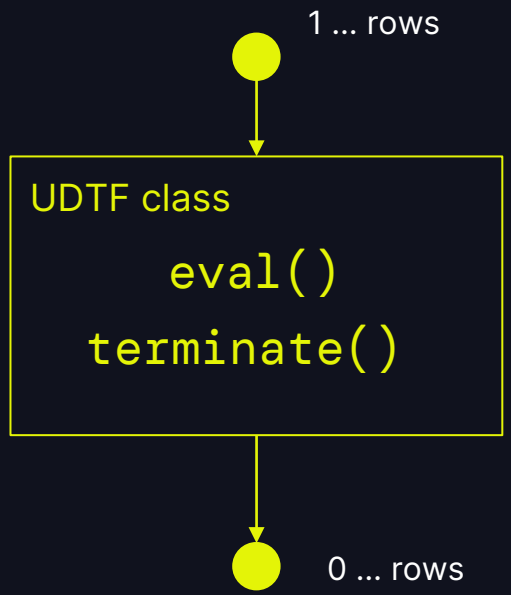
What is a Table Function?

- Definition
  - Table functions return a set of rows (a table) from a single input or a set of inputs.
- Comparison with Scalar Functions
  - Scalar functions return a single value, whereas table functions return multiple rows.
- Use Cases
  - Generating rows from a single value (e.g., expanding a range of numbers)
  - Complex data processing that returns multiple rows from complex operations
- Example
  - `range` function in SQL or DataFrame API
  - `SELECT * FROM range(1, 5)` or `spark.range(1, 5)` generates rows: 1, 2, 3, 4

What are UDTFs?

Available in Spark 3.5

- Definition
  - UDTFs (User-Defined Table Functions) in PySpark allow for custom data processing functions that return tables.
- Components:
  - UDTF class
  - `eval` function
  - `terminate` function



UDTF Class and eval Function

Standard UDTF

PYTHON

```python
from pyspark.sql.functions import udtf

@udtf(returnType="num: int, squared: int")
class SquareNumbers:

    def eval(self, start: int, end: int):
        for num in range(start, end + 1):
            yield (num, num * num)
```

UDTF Class and eval Function

Standard UDTF

PYTHON

```python
from pyspark.sql.functions import udtf

@udtf(returnType="num: int, squared: int")
class SquareNumbers:

    def eval(self, start: int, end: int):
        for num in range(start, end + 1):
            yield (num, num * num)
```

UDTF Class and eval Function

Standard UDTF

PYTHON

```python
from pyspark.sql.functions import udtf

@udtf(returnType="num: int, squared: int")
class SquareNumbers:

    def eval(self, start: int, end: int):
        for num in range(start, end + 1):
            yield (num, num * num)
```

UDTF Class and eval Function

Standard UDTF

PYTHON

```python
# DataFrame API
from pyspark.sql.functions import lit

SquareNumbers(lit(1), lit(3)).show()



# Use in SQL
spark.udtf.register("square_numbers", SquareNumbers)

spark.sql("SELECT * FROM square_numbers(1, 3)").show()
```

Capabilities of UDTFs

Available in Spark 3.5

Capabilities

- Custom data transformations
- Handling complex data processing logic

Use Cases

- Aggregating data
- Generating multiple output rows from a single input row

Areas for Improvement in UDTFs

Available in Spark 3.5

Areas for Improvement

- Fixed input schema and data types
- Static output schema

Challenges

- Adapting to varying data structures
- Extensibility for different use cases

POLYMORPHIC UDTFS

Introduction to Polymorphism

Available from Spark 4.0

Definition

- Polymorphism allows functions to handle different data types and structures.

Benefits

- Increased flexibility
- Enhanced reusability

Making UDTFs Polymorphic

Available from Spark 4.0

Steps to Achieve Polymorphism

- Removing the return type from the `@udtf` decorator
- Implementing the `analyze` static function
- Adapting to different input schemas

Benefits

- Enhanced flexibility
- Broader applicability

Making UDTFs Polymorphic

Polymorphic UDTF

PYTHON

```python
from pyspark.sql.types import StructType, StructField
from pyspark.sql.functions import udtf, AnalyzeArgument, AnalyzeResult

@udtf  # (returnType="num: int, squared: int")
class SquareNumbers:
    @staticmethod
    def analyze(start: AnalyzeArgument, end: AnalyzeArgument) -> AnalyzeResult:
        # Determine output schema based on input schema
        schema = StructType(
            [StructField("num", start.dataType), StructField("squared", start.dataType)])
        return AnalyzeResult(schema)

    def eval(self, start: Any, end: Any):
```

Making UDTFs Polymorphic

AnalyzeArgument and AnalyzeResult

- `AnalyzeArgument`
  - Used as argument types in the `analyze` function
  - Provides metadata about the input arguments
    - `dataType`, `value`, `isTable`, and `isConstantExpression`
- `AnalyzeResult`
  - Returned by the `analyze` function
  - Contains `schema`
  - Optionally includes:
    - `withSinglePartition`, `partitionBy`, `orderBy`, and `select`

DEMO

Conclusion

Summary

- UDTFs enable custom data transformations in PySpark
- Polymorphism enhances flexibility and reusability

Key Takeaways

- Understanding UDTFs and their capabilities
- Making UDTFs polymorphic for broader applicability