# SQL & Table API Ecosystem

Apache Flink SQL Training

https://github.com/ververica/sql-training


© 2019 Ververica

# Benefits of using SQL

- Schema-awareness at all times
  - Including NULL support
- Smooth integration with catalogs, connectors, formats
  - Hive catalog integration coming soon
- Large set of built-in operations and functions
- Use the same query to process streaming and batch data

© 2019 Ververica

# When not to use SQL

- Application that require evolution
  - Flink provides fault-tolerance and savepoint compatibility for SQL queries in the same version
  - Savepoint compatibility when migrating to a newer Flink version is not guaranteed yet
  - Modified queries cannot be loaded from a previous savepoint
- Applications that require access to state and time
  - Custom triggers and timers
  - Fine-grained state management

© 2019 Ververica

# Embedding SQL & Table API in DataStream Programs

# Embedding SQL in the DataStream API

```
val sensorData: DataStream[(String, Long, Double)] = ???

// register DataStream
tableEnv.registerDataStream("sensorData", sensorData, 'location, 'rowtime, 'tempF)

// query registered Table
val avgTempCTable: Table = tableEnv.sqlQuery("""
    SELECT
        TUMBLE_START(TUMBLE(time, INTERVAL '1' DAY) AS day,
        location,
        AVG((tempF - 32) * 0.556) AS avgTempC
    FROM sensorData
    WHERE location LIKE 'room%'
    GROUP BY location, TUMBLE(time, INTERVAL '1' DAY)
  """)

// go back to DataStream API
val avgTempC: DataStream[Row] = avgTempCTable.toAppendStream[Row]
```

© 2019 Ververica

# Embedding Table API in the DataStream API

```
val sensorData: DataStream[(String, Long, Double)] = ???

// convert DataStream into Table
val sensorTable: Table = sensorData.toTable(tableEnv, 'location, 'rowtime, 'tempF)

// define query on Table
val avgTempCTable: Table = sensorTable
  .window(Tumble over 1.day on 'rowtime as 'w)
  .groupBy('location, 'w)
  .select('w.start as 'day, 'location, (('tempF.avg - 32) * 0.556) as 'avgTempC)
  .where('location like "room%")

// go back to DataStream API
val avgTempC: DataStream[Row] = avgTempCTable.toAppendStream[Row]
```

© 2019 Ververica

# Ingesting a DataStream from a Table

```
val tEnv = TableEnvironment.getTableEnvironment(env)
tEnv.registerFunction("myParser", new MyParser())


// configure your data source and register as a table
tEnv
  .connect(new Kafka().topic("MyTopic") …)
  .withFormat(new Json().deriveSchema())
  .withSchema(
    new Schema()
      .field("name", Types.STRING)
      .field("prefs", Types.STRING))
  .registerTableSource("customers")

// define your table program
val table: Table = tEnv.sqlQuery("SELECT LOWER(name), myParser(prefs) FROM customers")
val table: Table = tEnv.scan("customers").select('name.lowerCase(), myParser('prefs))

// convert
val ds: DataStream[Customer] = table.toAppendStream[Customer]
```

© 2019 Ververica

# Defining Source & Sink Tables

# Defining External Tables

- Tables are always stored in external systems
  - Systems have different properties and schema definitions
- Map schemas from external systems to Flink and vice-versa

**Java/Scala**

```
tableEnvironment
  .connect(...)
  .withFormat(...)
  .withSchema(...)
  .inAppendMode()
  .registerTableSink("MyTable")
```

**YAML**

```
name: MyTable
type: sink
update-mode: append
connector: ...
format: ...
schema: ...
```

See also: https://ci.apache.org/projects/flink/flink-docs-stable/dev/table/connect.html

© 2019 Ververica

# Connect to Storage Systems - Connector

## Java/Scala

```
.connect(
  new Kafka()
    .version("0.10")
    .topic("test-input")
    .startFromEarliest()
    .property(
      "zookeeper.connect",
      "localhost:2181")
    .property(
      "bootstrap.servers",
      "localhost:9092")
)
```

## YAML

```
connector:
  type: kafka
  version: "0.10"
  topic: test-input
  startup-mode: earliest-offset
  properties:
    - key: zookeeper.connect
      value: localhost:2181
    - key: bootstrap.servers
      value: localhost:9092
```

© 2019 Ververica

# Connect to Storage Systems - Format

## Java/Scala

```
.withFormat(
  new Avro()
    .avroSchema("""
      {
        "namespace": "org.myorganization",
        "type": "record",
        "name": "UserMessage",
        "fields": [
          {"name": "user", "type": "long"},
          {"name": "message", "type": "string"}
        ]
      }""")
)
```

## YAML

```
format:
  type: avro
  avro-schema: >
    {
      "namespace": "org.myorganization",
      "type": "record",
      "name": "UserMessage",
      "fields": [
        {"name": "ts", "type": "string"},
        {"name": "user", "type": "long"},
        {"name": "message", "type": "string"}
      ]
    }
```

© 2019 Ververica

# Supported Connectors and Formats in Flink 1.7

- Connectors:
  - Kafka (starting from 0.8)
  - Elasticsearch 6
  - Filesystem

- Formats:
  - JSON
  - Avro
  - CSV

- API for custom connectors and formats (Java/Scala and SQL Client)

© 2019 Ververica

# Hands On Exercises

# Running Queries on Streams

Continue with Session 6
"Writing Query Results to External Tables"
in the SQL training wiki:

https://github.com/ververica/sql-training/wiki

We are here to help!

© 2019 Ververica


www.ververica.com

@VervericaData