Cross-Language JdbcIO Enabled By Beam Portable Schemas
Agenda

- **JdbcIO**
  - Java Database Connectivity
  - Beam's JdbcIO

- **Portable Beam Schemas**
  - Types across language
  - Schema translation
  - Types and coders
  - Portable logical types

- **Python xlang JdbcIO**
  - use cases
  - run on different runners
  - deal with unsupported types
Java Database connectivity (JDBC) is an API defining how a client may access a database for Java.

- Standardized API (java.sql.*, javax.sql.*)
- JDBC driver (class) interact with a database
- Relational databases
One of the earliest IO connector in Beam ([BEAM-244], 0.3.0-incubating)

In Java SDK `org.apache.beam.io.jdbc`

- `JdbcIO.read()` read from JDBC datasource
- `JdbcIO.write()` write to JDBC
- (since v2.32) `JdbcIO.readWithPartitions()` Parallel reading from a JDBC datasource
In Python SDK `apache_beam.io.jdbc` since v2.24 (BEAM-10135, BEAM-10136)

- `ReadFromJdbc`
- `WriteToJdbc`

In Go SDK .../beam/sdks/v2/go/pkg/beam/io/xlang/jdbcio/jdbc since v2.37 (Beam-13293)

- `Read`
- `Write`
Beam SDK prefers to follow the convention of that language

PCollection<user_type> -> PCollection[?]
“Beam schemas are a new and enhanced type system for Beam, making element structure explicit to support new concise transforms, relational-style optimization and execution, columnar optimization, and automatic type coercions.”

Provide cross-language support in every Beam SDK.
Portable beam schemas

Schemas define types, coders pack/unpack values

- **SDK schema**
- **proto schema**
  - **Schema translation**
  - **Values / Rows / User types**
    - **Coders**
    - **Stream**

Language implementation

Language independent
## Types and Coders: support status

### Primitive types

* [https://s.apache.org/beam-schemas](https://s.apache.org/beam-schemas)

<table>
<thead>
<tr>
<th>Type</th>
<th>BYTE</th>
<th>INT16</th>
<th>INT32</th>
<th>INT64</th>
<th>FLOAT</th>
<th>DOUBLE</th>
<th>STRING</th>
<th>BOOLEAN</th>
<th>BYTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>byte</td>
<td>short</td>
<td>int</td>
<td>long</td>
<td>float</td>
<td>double</td>
<td>String</td>
<td>bool</td>
<td>byte[]</td>
</tr>
<tr>
<td>Python</td>
<td>np.int8</td>
<td>np.int16</td>
<td>np.int32</td>
<td>np.int64, int</td>
<td>np.float32</td>
<td>np.float64, float</td>
<td>str</td>
<td>bool</td>
<td>bytes</td>
</tr>
<tr>
<td>Go</td>
<td>int8</td>
<td>int16</td>
<td>int32</td>
<td>int64</td>
<td>float32</td>
<td>float64</td>
<td>string</td>
<td>bool</td>
<td>[]byte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Java</th>
<th>Python</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>ByteCoder</td>
<td>BytesCoder</td>
<td><strong>currently (2.48) cast to int64 then encode</strong></td>
</tr>
<tr>
<td>Python</td>
<td>BigEndianShortCoder</td>
<td>BigEndianShortCoder (2.46+)</td>
<td>coder/varint</td>
</tr>
<tr>
<td>Go</td>
<td>VarIntCoder</td>
<td>VarIntCoder</td>
<td>coder/long</td>
</tr>
<tr>
<td></td>
<td>VarLongCoder</td>
<td>FloatCoder</td>
<td>coder/floating</td>
</tr>
<tr>
<td></td>
<td>FloatCoder</td>
<td>SinglePrecisionCoder (2.42+)</td>
<td>coder/float (2.42+)</td>
</tr>
<tr>
<td></td>
<td>DoubleCoder</td>
<td>coder/double</td>
<td>coder/stringutf8</td>
</tr>
<tr>
<td></td>
<td>StringUtf8Coder</td>
<td>coder/bool</td>
<td>coder/bytes</td>
</tr>
</tbody>
</table>
How to handle SQL types VARCHAR(size), BINARY(size), DECIMAL(size, digits), TIMESTAMP

- **Non-portable logical types (Java SDK)**
  - Without a URN. Identifier like "VARCHAR", "VARBINARY"
  - Not recognized cross-lang (Error beam:logical_type:javasdk:v1) (Q, #19817, #23526)

- **Portable logical types**
  - identifier is a URN beam:logical_type:xxx:v1
  - A uniform representation type, and a language type per SDK
  - Arguments
Beam portable logical types

Example:

**micros_instant**

<table>
<thead>
<tr>
<th>Java SDK</th>
<th>Python SDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.time.Instant</td>
<td>beam.utils.timestamp.Time</td>
</tr>
</tbody>
</table>

Standard logical types

- URN defined in proto, supposed to be understood by all SDKs.
- Each SDK's LogicalType implementation defines language type and conversion rules.
Portable logical types: support status

**Example:**

**micros_instant**

<table>
<thead>
<tr>
<th>Type</th>
<th>Java/Python</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td>micros_instant</td>
<td>2.33</td>
<td></td>
</tr>
<tr>
<td>millis_instant</td>
<td>2.42</td>
<td></td>
</tr>
<tr>
<td>decimal</td>
<td>2.43</td>
<td>N/A</td>
</tr>
<tr>
<td>fixed_bytes</td>
<td>2.44</td>
<td></td>
</tr>
<tr>
<td>var_bytes</td>
<td>2.44</td>
<td></td>
</tr>
<tr>
<td>fixed_char</td>
<td>2.44</td>
<td></td>
</tr>
<tr>
<td>var_char</td>
<td>2.44</td>
<td></td>
</tr>
</tbody>
</table>

**Standard logical types**

- URN defined in proto, supposed to be understood by all SDKs.
- Each SDK's LogicalType implementation defines language type and conversion rules
Use case: xlang JdbcIO

SQL schema / Python NamedTuple

```python
JdbcWriteTestRow = typing.NamedTuple(  
    "JdbcWriteTestRow",  
    [
        ("f_id", int),
        ("f_real", float),
        ("f_fixedchar", str),
        ("f_varchar", str),
        ("f_bin", bytes),
        ("f_timestamp", Timestamp),
        ("f_decimal", Decimal)  
    ])
```

Write

```python
coders.registry.register_coder(JdbcWriteTestRow, coders.RowCoder)
with Pipeline(options=options) as p:
    input = p | SyntheticSource(...) | MapToRow(...)
    input | WriteToJdbc(  
        table_name=self.table_name,  
        driver_class_name=self.driver,  
        jdbc_url=self.jdbc_url.replace('localhost', 'host.docker.internal'),  
        username=self.username,  
        password=self.password))
```

Code available at https://gist.github.com/Abacn/3fa72fab4b0bbf5e3de395106ef47cfb
Use case: xlang JdbcIO

Read

```
LogicalType.register_logical_type(MillisInstant)
with Pipeline(options=options) as p:
    output = (p | 'Read from jdbc' >> ReadFromJdbc(
        table_name=self.table_name,
        driver_class_name=self.driver,
        jdbc_url=jdbc_url,
        username=username,
        password=password))
```

Partitioned read (since v2.46)

```
LogicalType.register_logical_type(MillisInstant)
with Pipeline(options=options) as p:
    input = (p | 'Read from jdbc' >> ReadFromJdbc(
        table_name=self.table_name,
        driver_class_name=self.driver,
        jdbc_url=jdbc_url,
        username=username,
        password=password,
        partition_column='f_id',
        partitions=100))
```
Read from Jdbc and write to BigQuery (storage write API, since 2.47.0)

LogicalType.register_logical_type(MillisInstant)
with Pipeline(options=options) as p:
    input = (p | 'Read from jdbc' >> ReadFromJdbc(...))
    _ = (input
        | beam.Map(lambda r: r.as_dict())
        | WriteToBigQuery(
            table=table,
            method=WriteToBigQuery.Method.STORAGE_WRITE_API,
            schema=SCHEMA))
What needed to make pipeline run?

- For all runners
  - Python environment with Beam installed
  - An Java environment - needs for expansion service

- Direct runner
  - Additionally, a docker environment - run container image for other SDK

- Portable runner (Spark/Flink/...)
  - docker environment also needed to run job server

* For released beam versions, expansion service jar are automatically downloaded first-time run. Containers are also pulled automatically
What if the logical type not supported?

- Solution 1: Implement the logical type in place
- Solution 2: cast to string
  - Example: Schema with DATE and TIME field
  - Gives non-portable DATE and TIME logical type in Java SDK
    ValueError: No logical type registered for URN 'beam:logical_type:javasdk:v1'

```python
rows = (p | 'Read from jdbc' >> ReadFromJdbc(
    query=f"select f_id, CAST(f_date as TEXT), CAST(f_time as TEXT), f_timestamp from {table_name}",
    table_name=table_name,
    driver_class_name=self.driver,
    jdbc_url=self.jdbc_url.replace('localhost', 'host.docker.internal'),
    username=self.username,
    password=self.password))
```
QUESTIONS?