// Licensed to the Apache Software Foundation (ASF) under one

// or more contributor license agreements. See the NOTICE file

// distributed with this work for additional information

// regarding copyright ownership. The ASF licenses this file

// to you under the Apache License, Version 2.0 (the

// "License"); you may not use this file except in compliance

// with the License. You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing,

// software distributed under the License is distributed on an

// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY

// KIND, either express or implied. See the License for the

// specific language governing permissions and limitations

// under the License.

#include "ipc.h"

#include <memory>

#include "arrow/compute/cast.h"

#include "arrow/python/pyarrow.h"

namespace arrow {

namespace py {

PyRecordBatchReader::PyRecordBatchReader() {}

Status PyRecordBatchReader::Init(std::shared\_ptr<Schema> schema, PyObject\* iterable) {

schema\_ = std::move(schema);

iterator\_.reset(PyObject\_GetIter(iterable));

return CheckPyError();

}

std::shared\_ptr<Schema> PyRecordBatchReader::schema() const { return schema\_; }

Status PyRecordBatchReader::ReadNext(std::shared\_ptr<RecordBatch>\* batch) {

PyAcquireGIL lock;

if (!iterator\_) {

// End of stream

batch->reset();

return Status::OK();

}

OwnedRef py\_batch(PyIter\_Next(iterator\_.obj()));

if (!py\_batch) {

RETURN\_IF\_PYERROR();

// End of stream

batch->reset();

iterator\_.reset();

return Status::OK();

}

return unwrap\_batch(py\_batch.obj()).Value(batch);

}

Result<std::shared\_ptr<RecordBatchReader>> PyRecordBatchReader::Make(

std::shared\_ptr<Schema> schema, PyObject\* iterable) {

auto reader = std::shared\_ptr<PyRecordBatchReader>(new PyRecordBatchReader());

RETURN\_NOT\_OK(reader->Init(std::move(schema), iterable));

return reader;

}

CastingRecordBatchReader::CastingRecordBatchReader() = default;

Status CastingRecordBatchReader::Init(std::shared\_ptr<RecordBatchReader> parent,

std::shared\_ptr<Schema> schema) {

std::shared\_ptr<Schema> src = parent->schema();

// The check for names has already been done in Python where it's easier to

// generate a nice error message.

int num\_fields = schema->num\_fields();

if (src->num\_fields() != num\_fields) {

return Status::Invalid("Number of fields not equal");

}

// Ensure all columns can be cast before succeeding

for (int i = 0; i < num\_fields; i++) {

auto& src\_type = src->field(i)->type();

auto& schema\_type = schema->field(i)->type();

if (!src\_type->Equals(schema\_type) && !compute::CanCast(\*src\_type, \*schema\_type)) {

return Status::TypeError("Field ", i, " cannot be cast from ",

src->field(i)->type()->ToString(), " to ",

schema->field(i)->type()->ToString());

}

}

parent\_ = std::move(parent);

schema\_ = std::move(schema);

return Status::OK();

}

std::shared\_ptr<Schema> CastingRecordBatchReader::schema() const { return schema\_; }

Status CastingRecordBatchReader::ReadNext(std::shared\_ptr<RecordBatch>\* batch) {

std::shared\_ptr<RecordBatch> out;

ARROW\_RETURN\_NOT\_OK(parent\_->ReadNext(&out));

if (!out) {

batch->reset();

return Status::OK();

}

auto num\_columns = out->num\_columns();

auto options = compute::CastOptions::Safe();

ArrayVector columns(num\_columns);

for (int i = 0; i < num\_columns; i++) {

const Array& src = \*out->column(i);

if (!schema\_->field(i)->nullable() && src.null\_count() > 0) {

return Status::Invalid(

"Can't cast array that contains nulls to non-nullable field at index ", i);

}

ARROW\_ASSIGN\_OR\_RAISE(columns[i],

compute::Cast(src, schema\_->field(i)->type(), options));

}

\*batch = RecordBatch::Make(schema\_, out->num\_rows(), std::move(columns));

return Status::OK();

}

Result<std::shared\_ptr<RecordBatchReader>> CastingRecordBatchReader::Make(

std::shared\_ptr<RecordBatchReader> parent, std::shared\_ptr<Schema> schema) {

auto reader = std::shared\_ptr<CastingRecordBatchReader>(new CastingRecordBatchReader());

ARROW\_RETURN\_NOT\_OK(reader->Init(parent, schema));

return reader;

}

Status CastingRecordBatchReader::Close() { return parent\_->Close(); }

} // namespace py

} // namespace arrow