Datafusion invariants change to support qualified column

In order to support referencing qualified columns in Datafusion queries, we need introduce some changes to the existing invariants design documented in https://docs.google.com/document/d/1Asnz29uUS1t60QNbNBU9SiME274rja-hcDvX_RDraFU/edit#heading=h.w9oks8pxurba.

Notation change and expansion

- Field or physical field (was metadata): the tuple name, `arrow::DataType` and nullability flag (a bool whether values can be null), represented in this document by `PF(name, type, nullable)`
- Logical field: Field with a relation name. Represented in this document by `LF(relation, name, type, nullable)`
- `Physical column: field name with an unique index. Represented in this document by PCol(index, name)`
- `Logical column: field name with relation name. Represented by LCol(relation, name)`
- `Projected plan: plan with projection as the root node`

Logical schema and physical schema

Logical schema is a schema associated with a logical plan. It is modeled as the DFSchema struct in Datafusion. Its fields are stored as a vector of logical fields.

Physical schema is a schema associated with a physical plan. It is modeled as the Arrow Schema struct in Datafusion. Its fields are stored as a vector of physical fields. It is used by both physical plan and record batch.

There are two main differences between logical and physical schemas:

1. Logical schema fields are uniquely identified by their (relation, name) tuples.
2. Physical schema fields do not have the concept of relation and it can contain fields with the same names.

Set equality/invariant

We say Set(vector1) === Set(vector2) when Sorted(vector1) === Sorted(vector2). In order words, vector1 and vector2 contain the same set of items, but these items could be stored in different orders.

For example:

- Set([F1, F2]) === Set([F2, F1])
- Set([F1, F2]) !== Set([F1, F2, F3])

Builder

A function that knows how to build a new logical plan from an existing logical plan and some extra parameters.

build(logical_plan, params…) -> logical_plan

Dropped invariant

Field names are unique

We have dropped this invariant because physical fields do not contain relation. The reason we don't want to include relation in physical fields is because inherently, relation should be treated as external metadata to a piece of data file like parquet, csv or json. In real world production systems, relations are tracked by external metadata systems like Hive metastore and Delta tables.

For example, consider we have a csv file `1.csv` tracked as table `customer` in a metadata store and we want to perform an ETL transformation on `customer` table to populate `customer_blocked` table and store the output as a csv file `2.csv`.

The physical schema for the resulting Arrow record batch from the following query:

`SELECT customer.id FROM customer WHERE blocked = true`

should contain the field name: `id` instead of `customer.id`. If we store the result in `2.csv` using `customer.id` as the field name, then when we load the `2.csv` file into memory as `customer_blocked` table, we will end with up a new field name with a redundant relation prefix: `customer_blocked.customer_blocked.id`. This is clearly not what one would expect.

Existing systems like MySQL, PostgreSQL and Spark also strip relation field/column qualifiers from query outputs. For example, running the query `SELECT test.id FROM test` in MySQL outputs:

```
-----
| id |
-----
| 1  |
| 2  |
-----
```

Note that users can always use explicit aliases `select customer.id as “customer.id”` to obtain the alternate behavior if they so desire.

Added invariant

(relation, name) tuples in logical fields and logical columns are unique

Every logical field’s (relation, name) in a schema MUST be unique. Every logical column’s (relation, name) in a logical plan MUST be unique.

This invariant guarantees that `SELECT t1.id, t2.id FROM t1 JOIN t2...` unambiguously selects the field t1.id and t2.id in a logical schema in the logical plane.

Responsibility

It is the logical builder and optimizer’s responsibility to guarantee this invariant, by erroring if the user’s statement violates it.

Validation

Builder and optimizer MUST error if this is invariant violated on any logical node that creates a new schema (e.g. scan, projection, aggregation, join, etc.).

Changed/updated invariant

The physical schema is invariant under planning (3.5)

The physical schema derived by a physical plan returned by the planner MUST be equivalent to the physical schema derived by the logical plan passed to the planner. I.e.

```
plan(logical_plan).schema === logical_plan.physical_schema
```

Logical plan's physical schema is defined as logical schema with relation qualifiers stripped for all logical fields:

logical_plan.physical_schema = vector[strip_relation(f) for f in logical_plan.logical_fields]

This is used to ensure that the physical schema of its (logical) plan is what it gets in record batches, so that users can rely on the optimized logical plan to know the resulting physical schema.

Note that since a logical plan can be as simple as a single projection with a single function, "Projection f(c1,c2)", a corollary of this is that the physical schema of every "logical function -> physical function" must be invariant under planning.

Responsibility

Developers of physical and logical plans and planners MUST guarantee this invariant for every triplet (logical plan, physical plan, conversion rule).

Validation

Planners MUST validate this invariant, and in particular return an error when, during planning, a physical function's derived schema does not match the logical functions' derived schema.

The output schema equals the physical plan schema (3.6)

The schema of every RecordBatch in every partition outputted by a physical plan MUST be equal to the schema of the physical plan. Specifically,

```
physical_plan.evaluate(batch).schema = physical_plan.schema
```

Together with other invariances, this ensures that the consumers of record batches do not need to know the output schema of the physical plan; they can safely rely on the record batch's schema to perform downscaling and naming.

Responsibility

Physical nodes MUST guarantee this invariant.

Validation

Execution Contexts CAN validate this invariant.

Logical schema is set invariant under logical optimization for non-projected plan (3.7)

The logical schema derived by a non-projected logical plan returned by the logical optimizer MUST be set equivalent to the schema derived by the logical plan passed to the planner:

```
Set(optimize(logical_plan).schema) === Set(logical_plan.schema)
```

This is used to ensure that plans can be optimized without jeopardizing future referencing columns (name) or assumptions about their schemas. By relaxing the equality restriction to set

equality, we give the optimizer extra room for optimization by reordering fields. For example, switching the join side in JOIN optimizer.

Responsibility

Logical optimizers MUST guarantee this invariant.

Validation

Users of logical optimizers SHOULD validate this invariant.

Logical schema is invariant under logical optimization for projected plan (3.7)

The logical schema derived by a projected logical plan returned by the logical optimizer MUST be equivalent to the logical schema derived by the logical plan passed to the planner:

```
optimize(logical_plan).schema === logical_plan.schema
```

This is used to ensure that plans can be optimized without jeopardizing future referencing logical columns (name and index) or assumptions about their schemas. Enforcing strict equality for projected plans gives users the ability to control field orders using projections. One of such use-cases would be accessing columns with duplicate names in resulting record batches by index.

Responsibility

Logical optimizers MUST guarantee this invariant.

Validation

Users of logical optimizers SHOULD validate this invariant.

Physical schema is set invariant under physical optimization for non-projected plan (3.8)

The physical schema derived by a non-projected physical plan returned by the physical optimizer MUST be set equivalent to the physical schema derived by the physical plan passed to the planner:

```
Set(optimize(physical_plan).schema) === Set(physical_plan.schema)
```

This is used to ensure that physical plans can be optimized without jeopardizing assumptions about their schema. By relaxing the equality restriction to set equality, we give the optimizer extra room for optimization by reordering fields. For example, switching the join side in JOIN optimizer.

Responsibility

Optimizers MUST guarantee this invariant.

Validation

Users of optimizers SHOULD validate this invariant.

Physical schema is invariant under physical optimization for projected plan (3.8)

The physical schema derived by a projected physical plan returned by the physical optimizer MUST match the physical schema derived by the physical plan passed to the planner:

```
optimize(physical_plan).schema === physical_plan.schema
```

This is used to ensure that plans can be optimized without jeopardizing future referencing logical columns (name and index) or assumptions about their schemas. Enforcing strict equality for projected plans gives users the ability to control field orders using projections. One of such use-cases would be accessing columns with duplicate names in resulting record batches by index.

Responsibility

Optimizers MUST guarantee this invariant.

Validation

Users of optimizers SHOULD validate this invariant.