Overview - Implementation

- New rules use isolated unit testing by running a heuristic optimizer with a single rule on a handcrafted plan.
- Code quality is good overall, documentation can be improved and better testing for rules (old and new) should be implemented.
- Plan quality has been manually evaluated by comparing to Umbra.
Overview - Goals

75%: Add basic logical → logical rules and support partial exploration.

100%: Multi-stage optimizer (rewriting + heuristic rule wrapper), add advanced logical → logical rules, improve testing.

125%: Support unnesting arbitrary queries, add physical properties.
Overview - Progress

- Partial exploration is supported.
- Multi-stage optimizer is fully supported.
- Many new logical→logical rules have been added, including filter pushdown and partial projection pushdown.
- Isolated unit testing has been introduced.
- Many bugs have been fixed throughout the semester.
- Physical Properties and Subquery unnesting are a work in progress.
Projection Pushdown

This PR implements a part of the projection transpose series of rules. It also includes a fair amount of refactoring.

Projection Merge Rule

- This rule matches on two projection nodes and combines the two nodes into one.
- It is added to the heuristic optimizer pass before the cascades optimizer. In the future, it should also be added to a pass after the cascades optimizer.

Projection Filter Transpose Rule

- This rule matches pushes a projection node passed a filter node. If the filter node contains columns that are not in this projection node, the top most projection node is also kept.
- It is added as a cascades rule.

Refactoring

Relevant functions for projection transpose rules can be found in `project_transpose_common.rs`. Rules are implemented in separate files as a part of the `projection_transpose` module rather than in all one file. Similarly, `FilterProjectTransposeRule` and `ProjectionPullUpJoin` were moved into this module.

Testing

Unit tests using the dummy heuristic optimizer were implemented.
Projection Pushdown

**List of Necessary Fixes Prior to Merge**

- Projection Push Down Join Rule can still apply on the top-level join, even if it’s just generating redundant projection nodes beneath the join node. Need to find a way to avoid applying the rule in cascade’s core logic (and in heuristic code path).
- Join enumeration is not possible with cyclic memo nodes. Need to implement a better join enumeration algorithm that is not brute forcing through everything in the memo table.
- Need a way to match on Scan nodes. Currently, ProjectRemoveRule is only possible in the Heuristic Optimizer pass because of this.
- Projection Agg Transpose Rule remains unimplemented.

**Projection Push Down Join Rule**

- This rule pushes a projection past a join node. It may still have a top most projection node, and in most cases creates a projection node above the left join child and a projection node above the right join child.
- This rule is commented out but intended to be a heuristic wrapper rule

**Projection Remove Rule**

- This rule matches on a projection node followed by a scan node.
- It is added in the preliminary heuristic pass

**Testing**

Unit tests using the dummy heuristic optimizer were implemented.
Physical Properties Framework - Design

- All expressions within one group are logically equivalent.
Physical Properties Framework - Design

- What if we have physical requirements? Eg. Column Order, Data Distribution

- One group have multiple subgroups, exprs within each subgroup satisfy the same physical property requirement
- Each subgroup has a winner
- Subgroup 0 has all expressions and has no physical property requirement
- Children are represented as (GroupId, SubGroupId)
Physical Properties Framework - Design
Physical Properties Framework - Design
Physical Properties Framework - Design
Physical Properties Framework - Design
Physical Properties Framework - Design

- In OptimizeInputTask, Physical properties requirement are separated to two parts: `handled_by_enforcer, handled_by_child`
- Three situations:
  - Can provide by expr itself, eg. SortMergeJoin
  - Cannot provide cannot pass down, eg. HashJoin, Union
  - Cannot provide can pass down, eg. Projection, Filter
- For multiple physical properties, we provide a framework to traverse all the possible combinations the registered physical properties provide to separate their requirements into two parts.
- Change 30% of the optd-core, need everybody’s agreement to make it nice and clean
Unnesting Arbitrary Queries - Status

- Somewhere between a proof of concept and a draft—work still heavily in progress.
- Parses and fully unnests a subset of correlated and uncorrelated subqueries.
- TODO
  - Formal testing and bugfixes
  - EXISTS clauses, IN clauses, ANY/ALL
  - Correctness issue with COUNT(*) (requires adding left outer join to plan)
  - Move some/all of this to rewriting stage to support multiple subqueries/ordering operations
  - “Sideways information passing” (subplans are duplicated now instead of making a DAG)
  - Optimizations from the paper are all missing
  - Pushing dependent joins past regular joins
Unnesting Arbitrary Queries - Demo

Parsed by DataFusion,
Converted to optd

Dependent join eliminated

Executed by DataFusion
Future Work

- Supporting Anti Join + Semi Join
- Rule Priorities
- Physical Properties Implementation
- Verify opt-d Correctness
- Projection Transpose Rules
- Filter Pull Up Rules
- Unnesting Arbitrary Subqueries