1. For filter push down, the optimizer needs to know which predicates are remaining. If there are remaining predicates, the optimizer will create a new Filter node. For project push down, optimizer also needs to know which columns are remaining. Because some table sources support nested project push down, but others don't.

2. all predicates != pushed predicates + remaining predicates.
   Table ParquetTableSource as an example: all pushed predicates should be retained, because Parquet filter is applied on RowGroup instead of each record.

3. In volcano planner, the rule will create a new TableScan node, and the TableScan should hold a new TableSource instance. In other words, we should create a new TableSouce after each pushdown.
   For example: project push down. If the origin table has fields a, b, c. The Project node has only one field: b. If the TableScan before pushdown and the TableScan after pushdown hold the same TableSource which schema is b. The pattern looks like:
   TableScan (a, b, c) -> TableSource (b)
   TableScan (b) -> TableSource (b)

   If the planner chooses the filter plan for some reason, the result is wrong. The correct pattern should be:
   TableScan (a, b, c) -> TableSource (a, b, c)
   TableScan (b) -> TableSource (b)

4. The TableScan node should generate a new digest for the new TableSource after pushdown, otherwise the optimizer may choose an unexpected plan.
   Optional options:
   a. generate digest through TableSource self (TableSource#explainSource): cons: each TableSource needs implement explainSource method
   b. generate digest through the pushed result. cons: TableSource should return the pushed predicates. pros: TableSource does not need explainSource method.

5. How to avoid the pushdown rules of infinite loop
   Optional options:
   a. use isFilterPushedDown flag
   b. use the result of getPushedFilters

6. If a TableSource is a FilterableTableSource and also an AggregateTableSource, PushFilterIntoScanRule needs to know whether the TableSource has pushed agg, PushAggregateIntoScanRule needs to know whether the TableSource has pushed filter. Because we can't push the filter into a TableSource which has pushed agg, and also can't push the agg into a TableSource which has pushed filter.
   Optional options:
   a. use isFilterPushedDown flag
   b. use the result of getPushedFilters

7. advance feature: TableSouce can return the cost based on the pushed filter or agg, the optimizer can choose the lowest-cost plan. (the pushdown plan is not necessarily best)

we can meet the requirement of ,4, 5, 6 at the same time through the result of pushdown.

```
public interface DynamicTableSource {
    DynamicTableSource copy();
    String asSummaryString();
}

// Option#1
public interface SupportsFilterPushDown {

        FilterPushdownResult applyFilters(List<ResolvedExpression> filters);

        final class FilterPushdownResult {
                final List<ResolvedExpression> remainingFilters;
                final List<ResolvedExpression> acceptedFilters;

                public FilterPushedResult(List<ResolvedExpression> remainingFilters,
List<ResolvedExpression> acceptedFilters) {
                        this.remainingFilters = remainingFilters;
                        this.acceptedFilters = acceptedFilters;
                }
        }

}

// Option#2
public interface SupportsFilterPushDown {

        List<ResolvedExpression> applyFilters(List<ResolvedExpression> filters);

        List<ResolvedExpression> getAcceptedFilters();
}


public interface SupportsProjectionPushDown {

  void applyProjection(TableSchema requiredSchema);

}

⇒

// T(a int, b ROW<f1 INT, f2 STRING>, c STRING)
// push projection (a int, b ROW<f2 STRING>)
public interface SupportsProjectionPushDown {

  // connectors can turn this off to disable nested projection,
  // then the {@code requiredSchema} will not prune nested fields,
  // many connectors don't support nested pushdown, e.g. Csv
  boolean supportsNestedProjectionPushedDown();

  /** @return the remaining schema after projection. */
  void applyProjection(TableSchema requiredSchema);

}
```

A simply example:

```
public class MyParquetTableSource implements DynamicTableSource, SupportsProjectionPushDown,
SupportsFilterPushDown {

        private FilterPredicate predicate;
        private TableSchema schema;
```

```
	private MyParquetTableSource(FilterPredicate predicate, TableSchema schema) {
		this.predicate = predicate;
		this.schema = schema;
	}

	@Override
	public DynamicTableSource copy() {
		return new MyParquetTableSource(predicate, schema);
	}

	@Override
	public String asSummaryString() {
		return "MySource";
	}

	@Override
	public PushedResult applyFilters(List<ResolvedExpression> filters) {
		// try to convert Flink filter expressions to Parquet FilterPredicates
		List<FilterPredicate> convertedFilters = new ArrayList<>();
		List<ResolvedExpression> acceptedFilters = new ArrayList<>();

		for (ResolvedExpression filter : filters) {
			FilterPredicate convertedPredicate = toParquetPredicate(filter);
			if (convertedPredicate != null) {
				acceptedFilters.add(filter);
				convertedFilters.add(convertedPredicate);
			}
		}

		// construct single Parquet FilterPredicate
		FilterPredicate parquetPredicate = null;
		if (!convertedFilters.isEmpty()) {
			// concat converted predicates with AND
			parquetPredicate = convertedFilters.get(0);

			for (FilterPredicate converted : convertedFilters.subList(1,
convertedFilters.size())) {
				parquetPredicate = FilterApi.and(parquetPredicate, converted);
			}
		}
		this.predicate = parquetPredicate;
		return PushedResult.of(filters, acceptedFilters);
	}

	// returns null if it is not supported
	private FilterPredicate toParquetPredicate(Expression exp) {
		// TODO
		return null;
	}

	@Override
	public boolean supportsNestedProjectionPushDown() {
		// it doesn't support nested fields push down
		return false;
	}

	@Override
	public void applyProjection(TableSchema requiredSchema) {
		// the pushed schema only prunes top-level columns, so apply it simply
		this.schema = requiredSchema;
	}
}
```