universität innsbruck

Architektur und Implementierung von Datenbanksystemen

Meilenstein 3

Team 3 - Gründlinger Diana, Huber Marcel, Klotz Thomas, Targa Aaron, Thalmann Matthias

Anforderungen

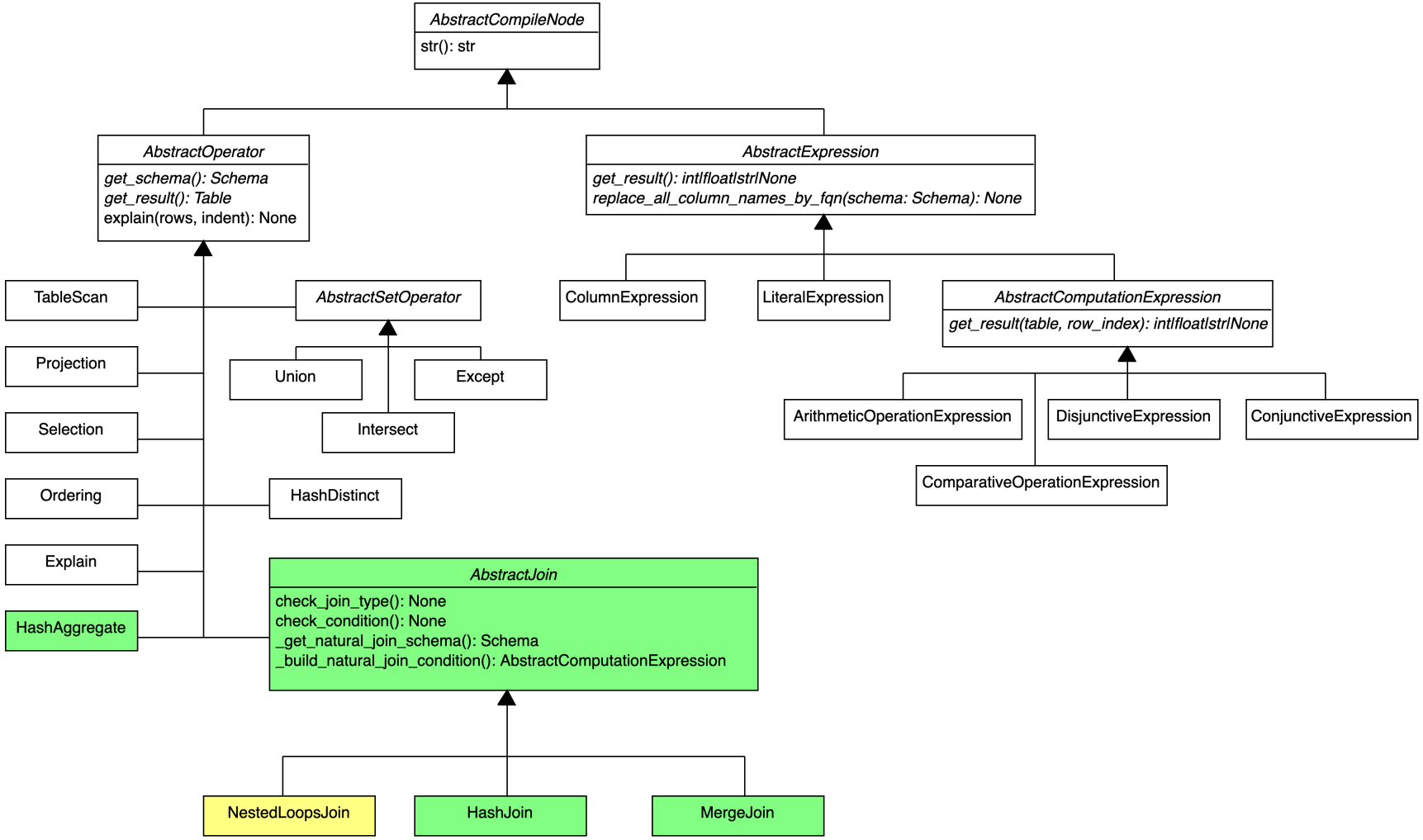
Operatoren:

- Join
- Grouping/Aggregation

Optimierungen:

- Selection Pushdown

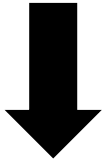
Operatoren


AbstractCompileNode
str(): str
AbstractOperator
get_schema(): Schema
get_result(): Table
explain(rows, indent): None
AbstractExpression
get_result(): int|float|str|None
replace_all_column_names_by_fqn(schema: Schema): None
TableScan
AbstractSetOperator
ColumnExpression
LiteralExpression
AbstractComputationExpression
get_result(table, row_index): int|float|str|None
Projection
Union
Except
Selection
Intersect
ArithmeticOperationExpression
DisjunctiveExpression
ConjunctiveExpression
Ordering
HashDistinct
ComparativeOperationExpression
Explain
AbstractJoin
check_join_type(): None
check_condition(): None
_get_natural_join_schema(): Schema
_build_natural_join_condition(): AbstractComputationExpression
HashAggregate
NestedLoopsJoin
HashJoin
MergeJoin

Hashjoin

```
explain pruefen join pruefen.VorlNr = hoeren.VorlNr hoeren;
```

```
+-----------------------------------------------------------------------------------------+
| Operator                                                                                |
|-----------------------------------------------------------------------------------------|
| -->NestedLoopsJoin(type=inner, natural=False, condition=(pruefen.VorlNr = hoeren.VorlNr)) |
| ---->TableScan(pruefen)                                                                 |
| ---->TableScan(hoeren)                                                                  |
+-----------------------------------------------------------------------------------------+
```

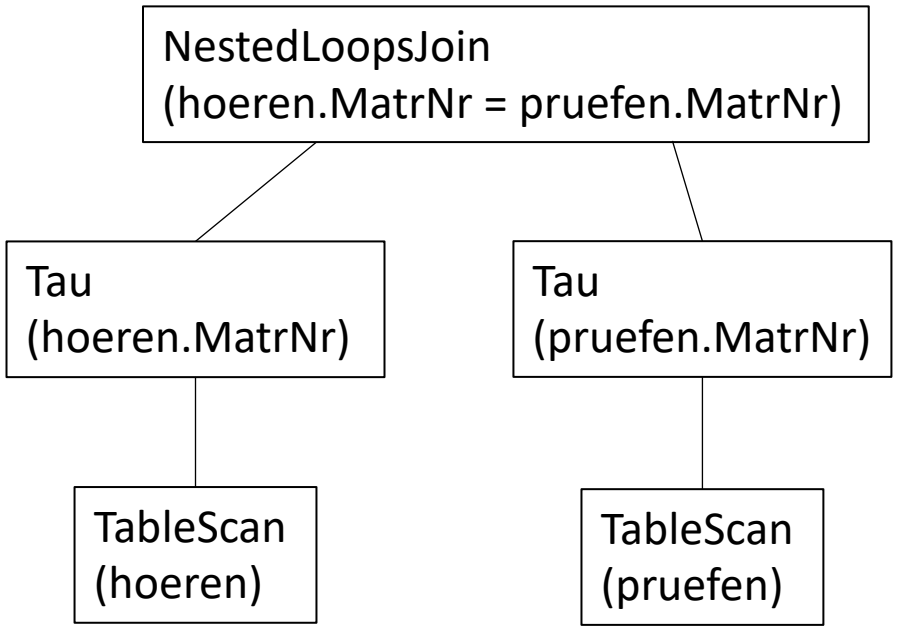


Optimizer:

If condition is equality or conjunction of equalities

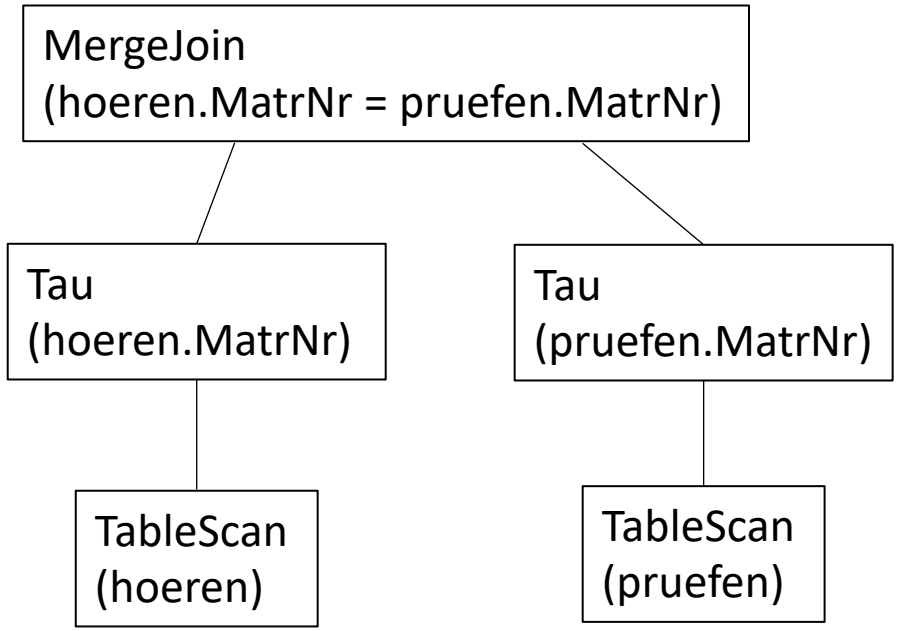
```
+-----------------------------------------------------------------------------------------+
| Operator                                                                                |
|-----------------------------------------------------------------------------------------|
| -->HashJoin(type=inner, natural=False, condition=(pruefen.VorlNr = hoeren.VorlNr))      |
| ---->TableScan(pruefen)                                                                 |
| ---->TableScan(hoeren)                                                                  |
+-----------------------------------------------------------------------------------------+
```

Mergejoin

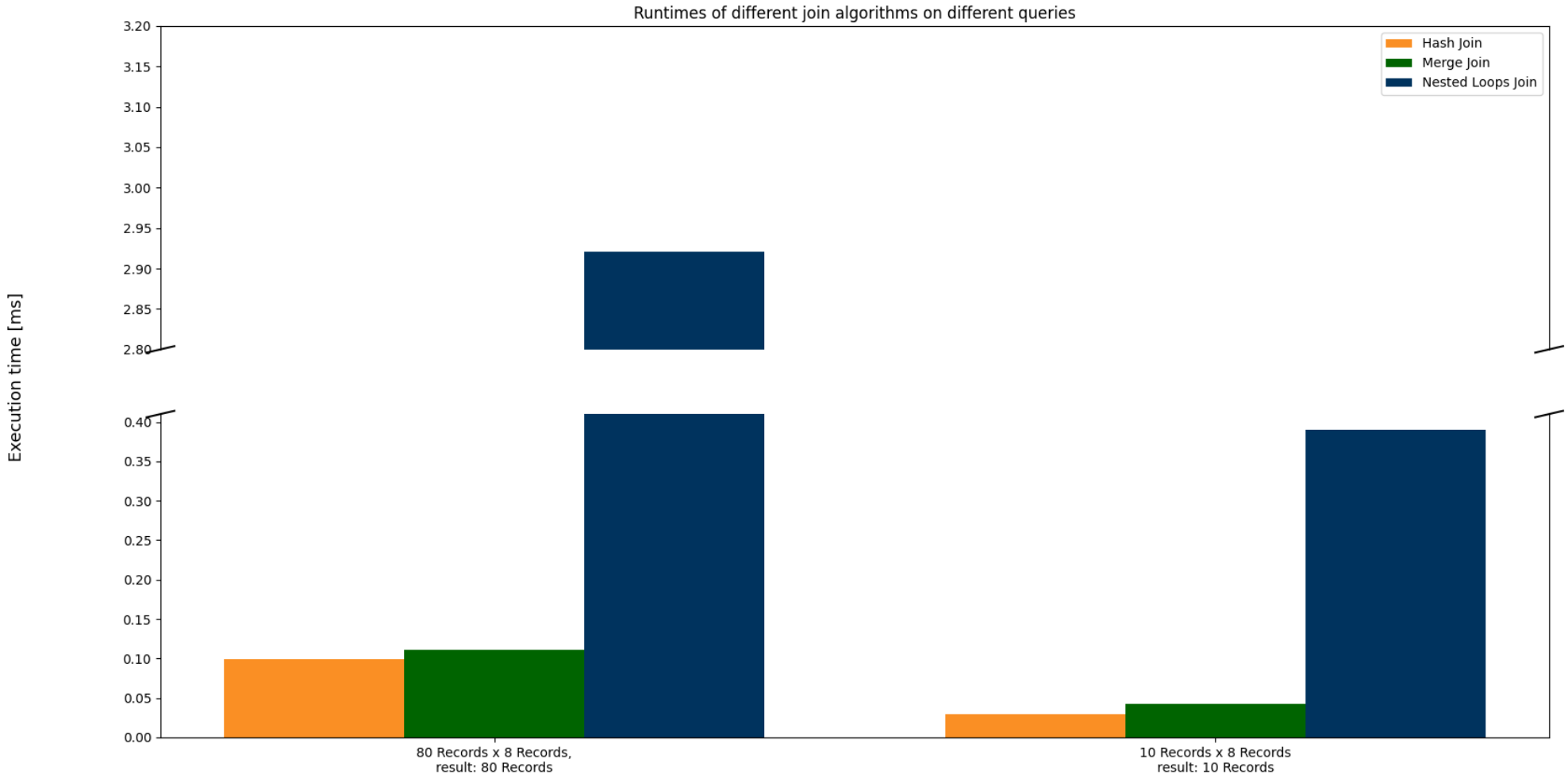


Optimizer If:

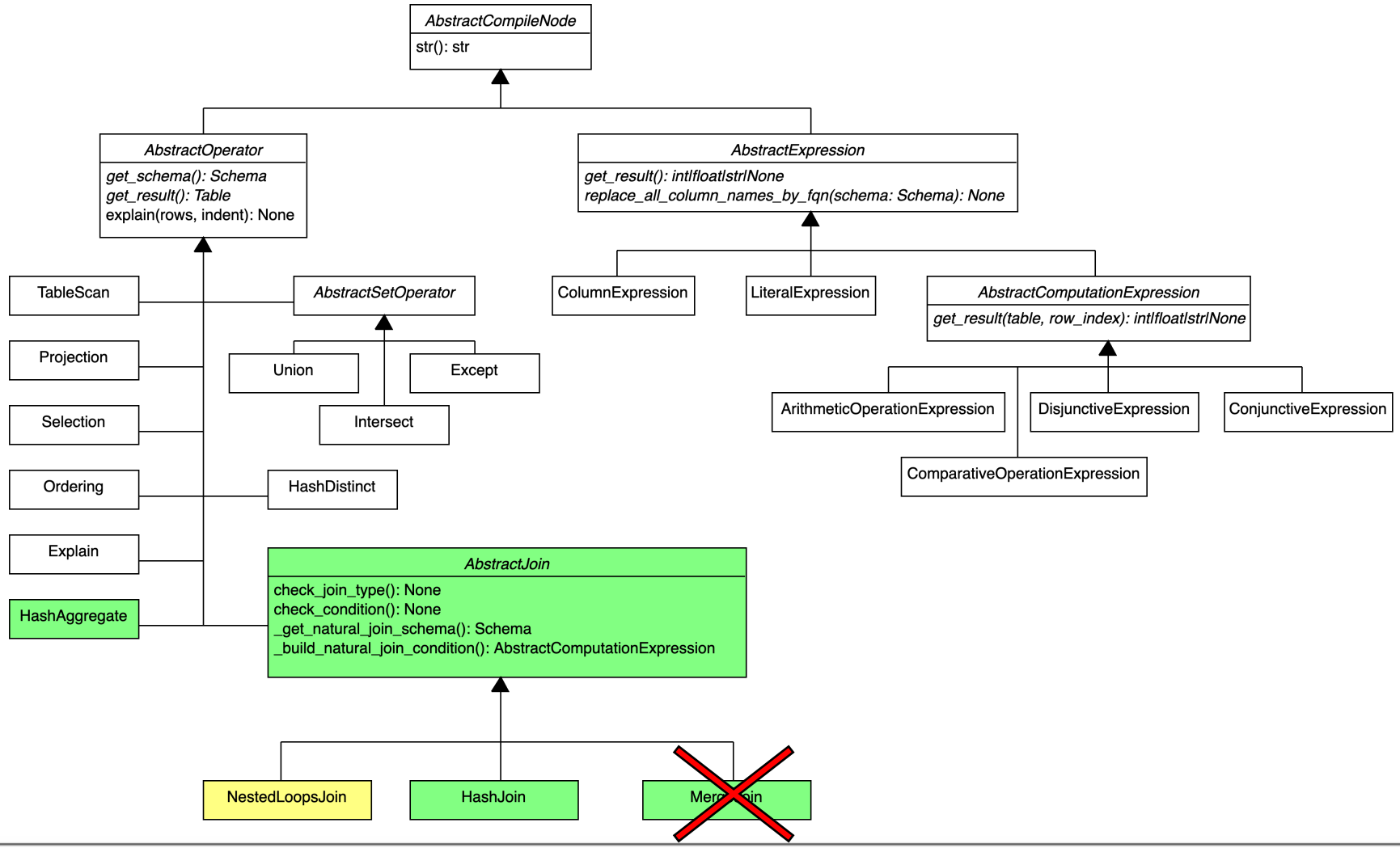
- Condition is equality or conjunction of equalities
- Condition columns are sorted



Performance



Operatoren


AbstractCompileNode
str(): str
AbstractOperator
get_schema(): Schema
get_result(): Table
explain(rows, indent): None
AbstractExpression
get_result(): intlfloatlstrlNone
replace_all_column_names_by_fqn(schema: Schema): None
TableScan
AbstractSetOperator
ColumnExpression
LiteralExpression
AbstractComputationExpression
get_result(table, row_index): intlfloatlstrlNone
Projection
Union
Except
Selection
Intersect
ArithmeticOperationExpression
DisjunctiveExpression
ConjunctiveExpression
Ordering
HashDistinct
ComparativeOperationExpression
Explain
AbstractJoin
check_join_type(): None
check_condition(): None
_get_natural_join_schema(): Schema
_build_natural_join_condition(): AbstractComputationExpression
HashAggregate
NestedLoopsJoin
HashJoin

Optimizer – Generell

1. Ausdrücke rekursiv vereinfachen
2. Konjunktionen in eigene „Selections“ auftrennen
3. „Selection Pushdown“ durchführen
4. Hintereinander liegende „Selections“ durch Konjunktionen zusammenführen

Optimizer – Vereinfachen

- Jede „Expression“ und jeder „Operator“ implementieren „simplify“-Methode

Beispiel: **ArithmeticOperationExpression**

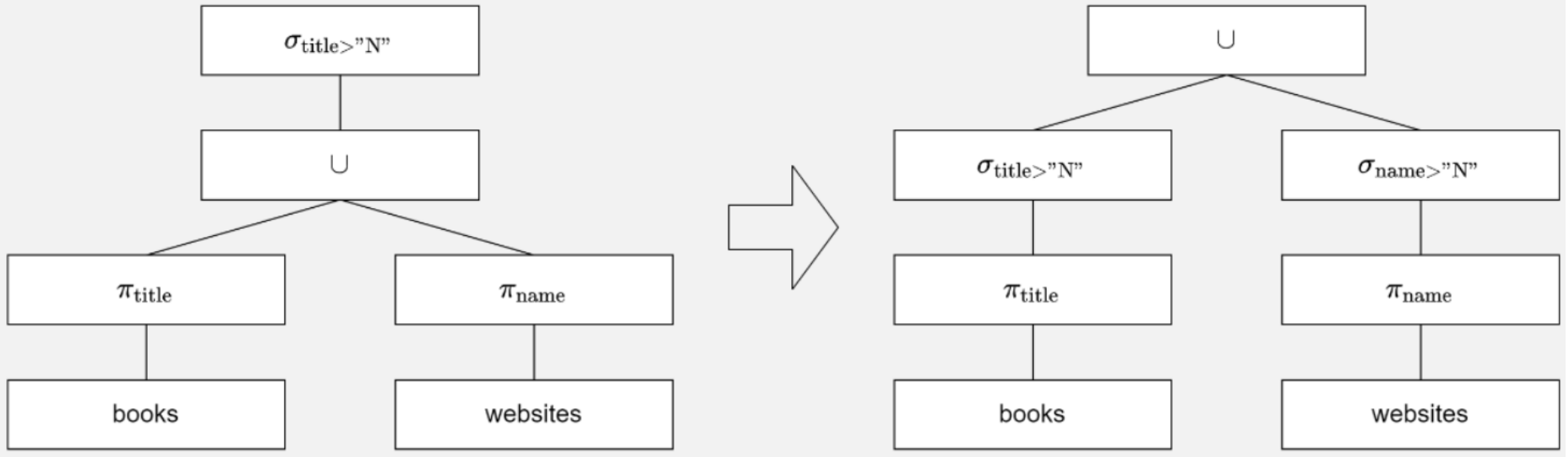
```
def simplify(self):
        """
        Tries to simplify all child-nodes and then itself.
        The resulting node is returned. If the current node is not replaced, itself is returned.
        """
        self.left = self.left.simplify()
        self.right = self.right.simplify()

        if isinstance(self.left, LiteralExpression) and isinstance(self.right, LiteralExpression):
            return LiteralExpression(self._compute(self.left.get_result(), self.right.get_result()))

        return self
```

Optimizer – Selection Pushdown

- Auf jede „Selection“ wird „_selection_push_down“ aufgerufen
- Funktion führt Set an bereits gepushten „Selections“
- „Selections“ werden mit „Distinct“-, „Ordering“- und „Selection“-Operatoren vertauscht
- Spezielle Regeln und Vorgehensweise bei „Projection“-, „Join“- und „Set“-Operatoren
- Über andere Operatoren kann nicht gepusht werden

Optimizer – Selection Pushdown – Set Operatoren

- „Selection“ wird auf linke Relation gepusht
- Spaltennamen werden nach rechter Relation umbenannt und auf sie gepusht

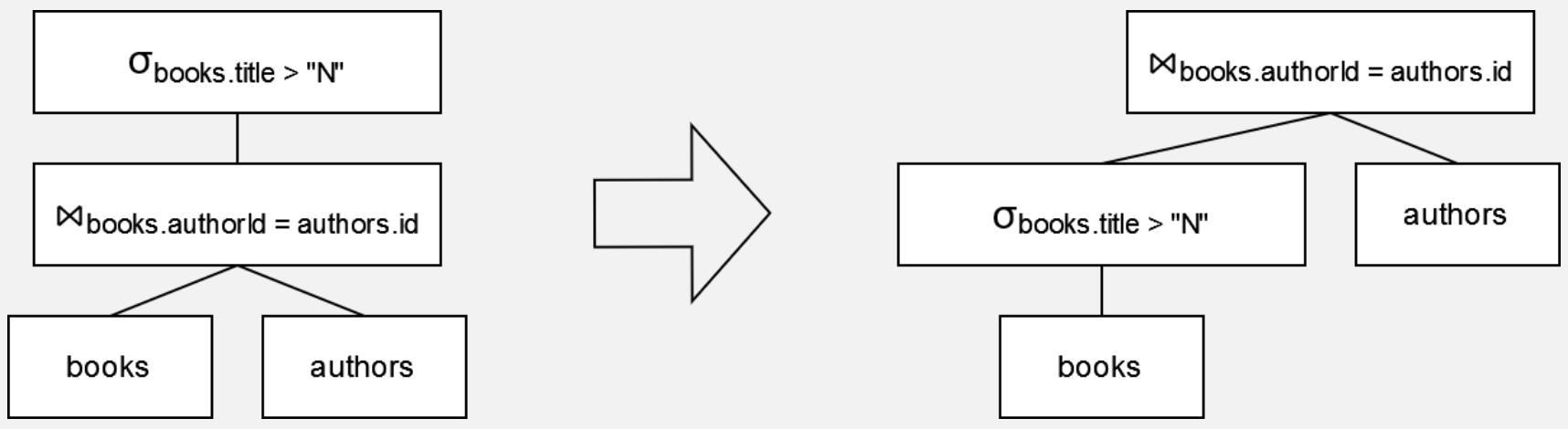


Quelle: https://git.uibk.ac.at/informatik/dbis/dbis-teaching/archimpl-course-material-2022/-/blob/main/slides/07_optimization1.pdf

Optimizer – Selection Pushdown – Join Operatoren

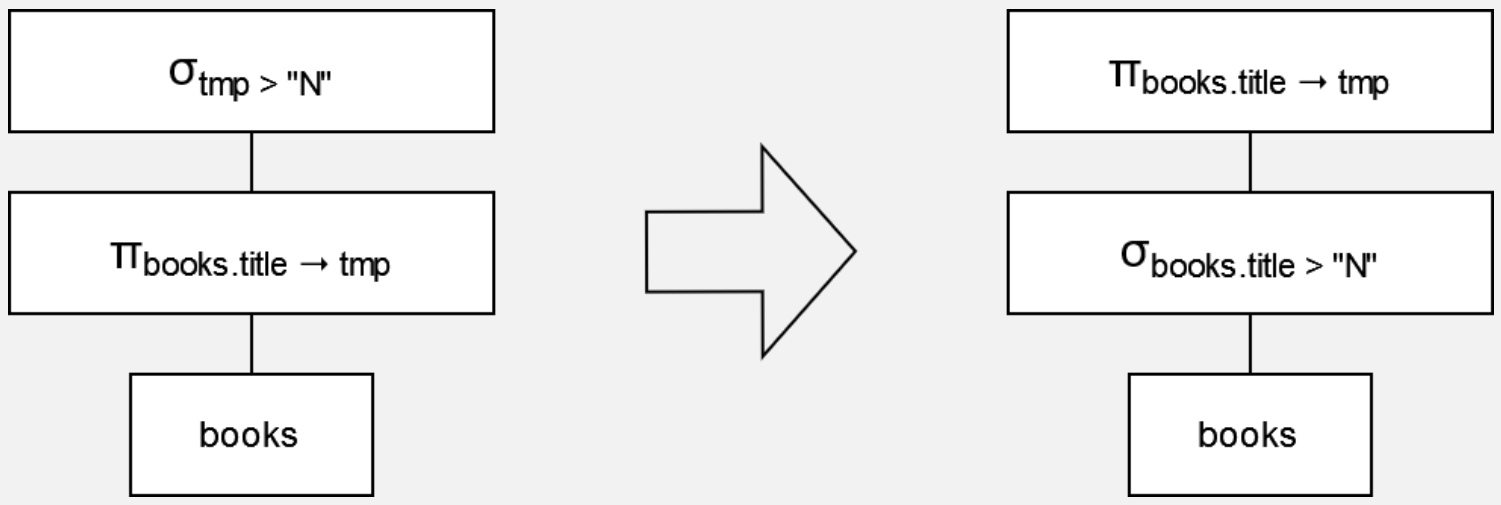
- Alle referenzierten Spalten dürfen in nur einer der Relationen vorkommen (Ausnahme: Natural Join)
- „Selection“ wird auf diese Relation gepusht
- Natural Join: Alle referenzierten Spalten müssen in beiden Relationen vorkommen

  → Hier wird auf beide Relationen gepusht (mit Umbenennung von Referenzen auf Spalten)

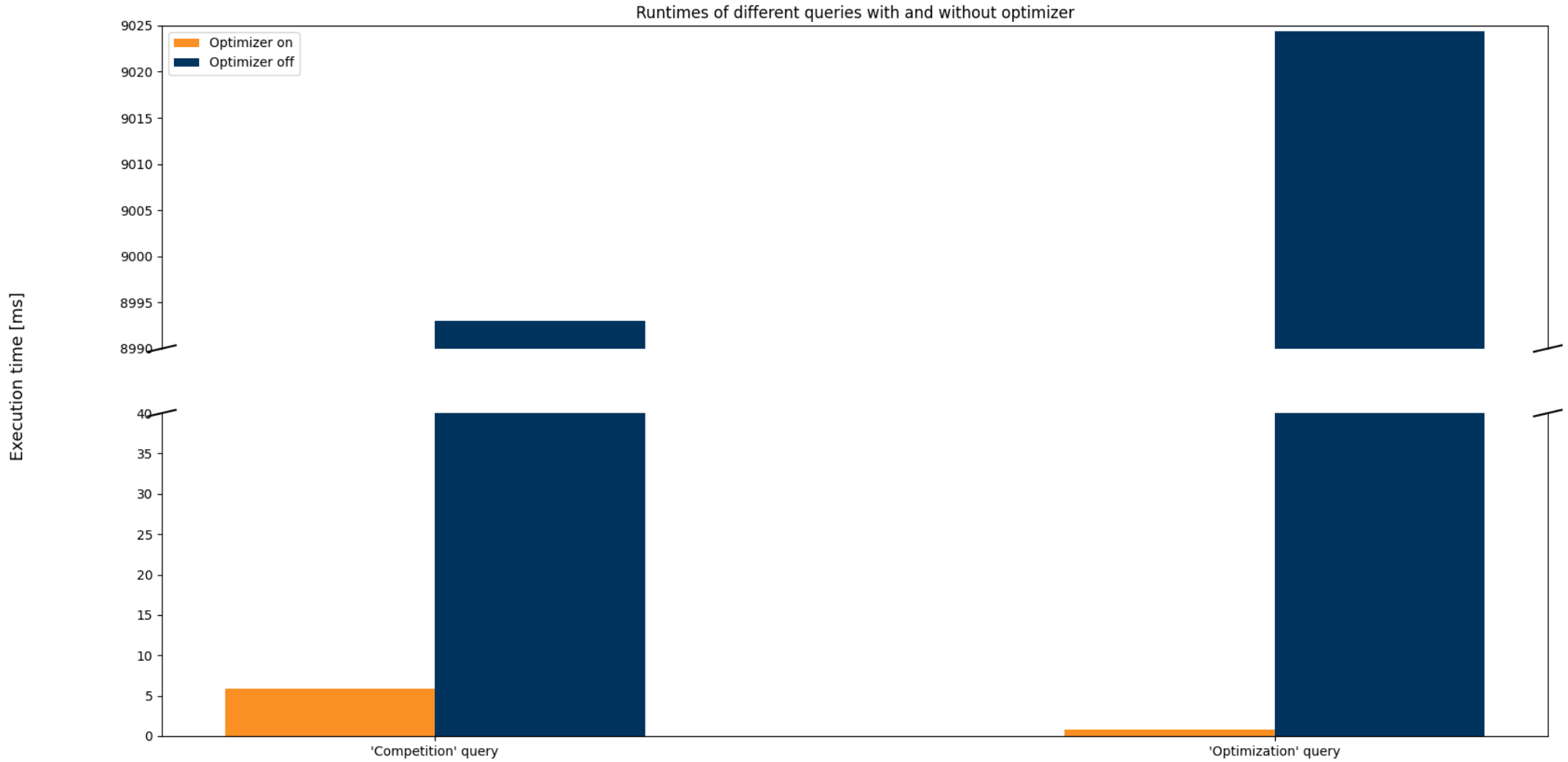


Optimizer – Selection Pushdown – Projection Operator

- Referenzierte Spalten müssen umbenannt werden
  → Aliase und Fully-Qualified-Namen
- Falls berechnete Spalten referenziert werden, kann nicht gepusht werden
  → z.B. bei `fullName = firstName + " " + lastName`
- Falls alle Bedingungen erfüllt werden, wird auf „Projection“ gepusht



Performance



www.uibk.ac.at