Scaling Database Systems
Winter Term 2025/26
Chair of Scalable Database Systems
Prof. Dr.-Ing. S. Scherzinger

UNIVERSITY OF PASSAU
Faculty of Computer Science and Mathematics

Milestone 2: Logical Optimization

Selection Pushing in Relational Algebra

In relational algebra, the following equivalencies apply (among others):

$$\sigma_{p_1 \wedge p_2 \wedge \cdots \wedge p_n}(R) = \sigma_{p_1}(\sigma_{p_2}(\ldots(\sigma_{p_n}(R))\ldots)) \tag{1}$$
$$\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R)) \tag{2}$$
$$\sigma_p(R_1 \times R_2) = \sigma_p(R_1) \times R_2 \tag{3}$$
$$\sigma_{R_1.A_1=R_2.A_2}(R_1 \times R_2) = R_1 \bowtie_{R_1.A_1=R_2.A_2} R_2 \tag{4}$$

Some remarks:

- Rule (1) states that a conjunction in a selection predicate may be broken into several nested selections. At the same time, nested selections may be merged into a single selection with a conjunctive predicate.
- Rule (2) states that nested selections may swap places.
- Rule (3) states that a selection can be pushed down over a cross product, if it only requires the attributes of one of the operands. In the rule as stated above, we assume that predicate p only requires attributes from R_1. The relation names and their attributes are available in the data dictionary.
- Rule (4) describes how a selection and a cross product may be merged into a theta-join, provided that the selection predicate is a join condition. This is the case if it compares attributes of R_1 and R_2 for equality.

Implement rule-based selection pushing and perform it on your relational algebra queries. Proceed in these phases:

(a) Complex selection predicates are broken up, according to rule (1).

(b) All selections are pushed down as far as possible, according to rules (2) and (3).

(c) Nested selections are merged again, according to rule (1).

(d) Joins are introduced, where possible, according to (4).

Note that there are more rules for the logical optimization of relational algebra, such as rules for join reordering or projection pushing. For *miniHive*, we restrict ourselves to this small set of optimization rules (for now).

Write a Python module `raopt.py` that takes a relational algebra query. You may assume that the query is the result of the canonical translation of SQL into relational algebra, so it uses only the operators σ, π, ρ, and $\times$. The data dictionary `dd` contains the relational schema and can be consulted during selection pushing.

Below is a sketch of how it should work.

```
>>> import radb.parse
>>> import raopt
>>>
>>> # The data dictionary describes the relational schema.
>>> dd = {}
>>> dd["Person"] = {"name": "string", "age": "integer", "gender": "string"}
>>> dd["Eats"] = {"name": "string", "pizza": "string"}
>>>
>>> stmt = """\project_{Person.name, Eats.pizza}
...             \select_{Person.name = Eats.name}(Person \cross Eats);"""
>>> ra = radb.parse.one_statement_from_string(stmt)
>>>
>>> ra1 = raopt.rule_break_up_selections(ra)
>>> ra2 = raopt.rule_push_down_selections(ra1, dd)
>>> ra3 = raopt.rule_merge_selections(ra2)
>>> ra4 = raopt.rule_introduce_joins(ra3)
>>>
>>> print(ra4)
\project_{Person.name, Eats.pizza} (Person \join_{Person.name = Eats.name}
 Eats)
```

Remarks: For Milestone 2, focus on *correctness* rather than efficiency.

Praktomat will execute the unit tests in `test_raopt.py` and `test_raopt_extended.py` upon submission. Solutions that rely on hard-coding or other non-general methods will not be accepted by us. Secret tests in Praktomat will detect such cases.

Submit your solution as a *single file* named `raopt.py` via Praktomat: https://praktomat.sdbs.fim.uni-passau.de (accessible only within the Uni Passau network or via the Uni Passau VPN). `raopt.py` should contain the following methods:

- `def rule_break_up_selections(ra): ...`
- `def rule_push_down_selections(ra, dd): ...`
- `def rule_merge_selections(ra): ...`
- `def rule_introduce_joins(ra): ...`

Deadline: December 9, 2025, 12:00 (noon, daylight). A valid submission must pass all public tests, the plagiarism check, and must not be hard-coded.
