Unnesting Arbitrary Queries

Thomas Neumann & Alfons Kemper

December 5, 2024

1 Preliminaries

(inner) join:r

$$T_1 \bowtie_p T_2 := \sigma_p(T_1 \times T_2)$$

dependent join:

$$T_1 \bowtie T_2 := \{t_1 \circ t_2 \mid t_1 \in T_1 \wedge t_2 \in T_2(t_1) \wedge p(t_1 \circ t_2)\}$$

Here the right hand side is evaluated for every tuple of the left hand side. We denote the attributes produced by an expression T by $\mathcal{A}(T)$, and free variables occurring in an expression T by $\mathcal{F}(T)$. To evaluate dependent join, $\mathcal{F}(T_2) \subseteq \mathcal{A}(T_1)$ must hold.

We use **natural join** in the join predicate to simplify the notation. We assume that all relations occuring in a query will have unique attribute names, even if they reference the same physical table, thus $A \bowtie B \equiv A \times B$. However, if we explicitly reference the same relation name twice, and call for the natural join, then the attribute columns with the same name are compared, and the duplicate columns are projected out. Consider, for example:

$$(A \bowtie C) \bowtie_{p \wedge \text{natural join } C} (B \bowtie C)$$

Here, the top-most join checks both the predicate p and compares the columns of C that come from both sides.

- **semi join**:

$$T_1 \ltimes T_2 := \{t_1 \mid t_1 \in T_1 \wedge \exists t_2 \in T_2 : p(t_1 \circ t_2)\}$$

- **anti semi join**:

$$T_1 \rhd_p T_2 := \{t_1 \mid t_1 \in T_1 \wedge \nexists t_2 \in T_2 : p(t_1 \circ t_2)\}$$

- **left outer join**:

$$T_1 \underset{p}{\bowtie} T_2 := (T_1 \bowtie_p T_2) \cup \{t_1 \circ_{a \in \mathcal{A}(T_2)} (a : null) \mid t_1 \in (T_1 \rhd_p T_2)\}$$

- **full outer join**:

$$T_1 \underset{p}{⟗} T_2 := (T_1 \underset{p}{\bowtie} T_2) \cup \{t_2 \circ_{a \in \mathcal{A}(T_1)} (a : null) \mid t_2 \in (T_2 \rhd_p T_1)\}$$

We define the dependent joins accordingly as $⋉$, $\rhd'$, $\bowtie'$, $⟗'$

group by:

$$\Gamma_{A;a:f}(e) := \{x \circ (a : f(y)) \mid x \in \prod_A (e) \wedge y = \{z \mid z \in e \wedge \forall a \in A : x.a = z.a\}\}$$

If groups its input e by A and evaluates one aggregation function.

map:

$$\chi_{a:f} := \{x \circ (a : f(x)) \mid x \in e\}$$

We define the attribute comparison operator $=_A$ as

$$t_1 =_A t_2 := \forall_{a \in A} : t_1.a = t_2.a$$

It compares NULL values as equal

2 Unnesting

The algebraic representation of a query with correlated subqueries results in a dependent join

2.1 Simple Unnesting

Consider

```
1  select ...
2  from lineitem l1 ...
3  where exists (select *
4                from lineitem l2
5                where l2.l_orderkey = l1.l_orderkey)
6  ...
```

This is translated into an algebra expression of the form

$$l_1 ⋉ (\sigma_{l_1.okey=l_2.okey}(l_2))$$

which is equivalent to

$$l_1 \ltimes_{l_1.okey=l_2.okey} (l_2)$$

2.2 General Unnesting

First, we translate the dependent join into a "nicer" dependent join (i.e., one that is easier to manipulate), and second, we will push the new dependent join down into the query until we can transform it into a regular join.

$$T_1 \bowtie_p T_2 \equiv T_1 \bowtie_{p \wedge T_1 =_{\mathcal{A}(D)} D} (D \bowtie T_2)$$

where

$$D := \prod_{\mathcal{F}(T_2) \cap \mathcal{A}(T_1)} (T_1)$$

In the original expression, we had to evaluate T_2 for every tuple of T_1. In the second expression, we first compute the domain D of all variables bindings, evaluate T_2 only once for every distinct variable binding, and then use a regular join to match the results to the original T_1 value. If there are a lot of duplicates, this already greatly reduces the number of invocations of T_2.

Consider the query for determining the best exam for every student.

$$\sigma_{e.grade=m}((\text{student } s \bowtie_{s.id=e.id} \text{exams } e)) \bowtie$$
$$(\Gamma_{\emptyset;m:min(e2.grade)})(\sigma_{s.id=e2.sid}\text{exams } e2)$$

The equivalence rule allows to restrict the computation of the best grades to each student

$$\ldots \prod_{d.id:s.id} ((\text{students } s \bowtie_{s.id=e.sid} \text{exams } e) \bowtie$$
$$(\Gamma_{\emptyset;m:min(e2.grade)}(\sigma_{d.id=e2.sid}\text{exams } e2)))$$

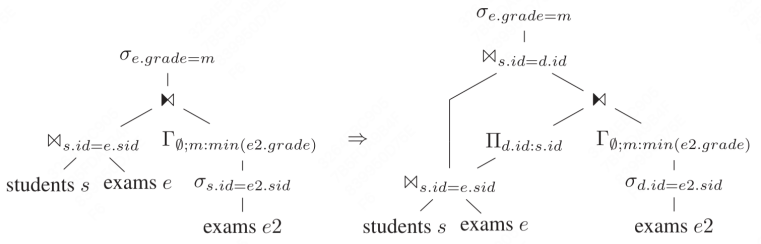

Figure 1: Example Application of Dependent Join "Push-Down"

Knowing that D contains no duplicates helps in moving the dependent join further down into the query. In the following, we will assume that any

relation named D is duplicate free, and in the following equivalences we only consider dependent joins where the left hand side is a set.

The untimate goal of our dependent join push-down is to reach a state where the right hand side no longer dependents on the left hand side, i.e.,

$$D \bowtie T \equiv D \bowtie T \text{ if } \mathcal{F}(T) \cap \mathcal{A}(D) = \emptyset$$

For selections, a push-down is very simple:

$$D \bowtie_{\sigma_p} (T_2) \equiv \sigma_p(D \bowtie T_2)$$

We first push the dependent join down as far as possible, until it can either be eliminated completely due to substitution, or until it can be transformed into a regular join. Once all dependent joins have been eliminated we can use the regular techniques like selection push-down and join reordering to re-optimize the transformed query.

$$D \bowtie (T_1 \bowtie_p T_2) = \begin{cases} (D \bowtie T_1) \bowtie_p T_2 & \mathcal{F}(T_2) \cap \mathcal{A}(D) = \emptyset \\ T_1 \bowtie_p (D \bowtie T_2) & \mathcal{F}(T_1) \cap \mathcal{A}(D) = \emptyset \\ (D \bowtie T_1) \bowtie_{p \wedge \text{natural join } D} (D \bowtie T_2) & \text{otherwise} \end{cases}$$

If we pushed the dependent join to both sides we have to augment the join predicate s.t. both sides are matched on the D values.

For *outer joins* we always have to replicate the dependent join if the inner side depends on it, as otherwise we cannot keep track of unmatched tuples from the outer side.

$$D \bowtie (T_1 ⟕_p T_2) \equiv \begin{cases} (D \bowtie T_1) ⟕_p T_2 & \mathcal{F}(T_2) \cap \mathcal{A}(D) = \emptyset \\ (D \bowtie T_1) ⟕_{p \wedge \text{natural join } D} (D \bowtie T_2) & \text{otherwise} \end{cases}$$

$$D \bowtie (T_1 ⟗_p T_2) \equiv (D \bowtie T_1) ⟗_{p \wedge \text{natural join } D} (D \bowtie T_2)$$

Similar for *semi join* and *anti join*:

$$D \bowtie (T_1 \ltimes_p T_2) \equiv \begin{cases} (D \bowtie T_1) \ltimes_p T_2 & \mathcal{F}(T_2) \cap \mathcal{A}(D) = \emptyset \\ (D \bowtie T_1) \ltimes_{p \wedge \text{natural join } D} (D \bowtie T_2) & \text{otherwise} \end{cases}$$

$$D \bowtie (T_1 \triangleright_p T_2) \equiv \begin{cases} (D \bowtie T_1) \triangleright_p T_2 & \mathcal{F}(T_2) \cap \mathcal{A}(D) = \emptyset \\ (D \bowtie T_1) \triangleright_{p \wedge \text{natural join } D} (D \bowtie T_2) & \text{otherwise} \end{cases}$$

group by

$$D \bowtie (\Gamma_{A;a:f}(T)) \equiv \Gamma_{A \cup \mathcal{A}(D);a:f}(D \bowtie T)$$

projection

$$D \bowtie (\Pi_A(T)) \equiv \Pi_{A \cup \mathcal{A}(D)}(D \bowtie T)$$

set operation

$$D \bowtie (T_1 \cup T_2) \equiv (D \bowtie T_1) \cup (D \bowtie T_2)$$
$$D \bowtie (T_1 \cap T_2) \equiv (D \bowtie T_1) \cap (D \bowtie T_2)$$
$$D \bowtie (T_1 \setminus T_2) \equiv (D \bowtie T_1) \setminus (D \bowtie T_2)$$

3 Problems

4 References

References