

# Paper Reading: Unnesting Arbitrary Queries

Presented by Shenghui Wu



# Introduction

- Unnesting arbitrary queries

# Preliminaries

- Regular (inner) join

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

- Dependent join (Apply)

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

- The attributes produced by an expression  $T$  :  $\mathcal{A}(T)$
- Free variables occurring in an expression  $T$  :  $\mathcal{F}(T)$
- To evaluate the dependent join ,  $\mathcal{F}(T_2) \subseteq \mathcal{A}(T_1)$  must hold.  
(the attributes required by  $T_2$  must be produced by  $T_1$ )

# Preliminaries

- Group by operator

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

- Map operator

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

- The attribute comparison operator  $=_A$

- This operator has *is* semantics, i.e., it compares NULL values as equal.

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

# Preliminaries

- For semi joins, anti joins, and outer joins ,we define the dependent variants accordingly.

⋈, ⋈<sub>l</sub>, ⋈<sub>r</sub>, ⋈<sub>o</sub>

- We assume that all relations occurring in a query will have unique attribute names, even if they reference the same physical table.

# Unnesting

- Push-down dependent join
- The ultimate goal

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

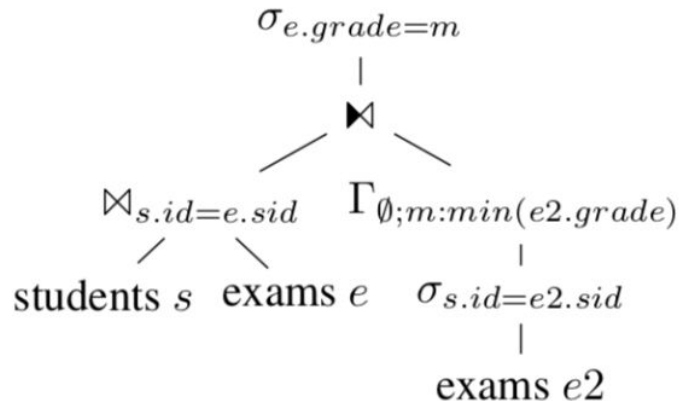
- General equivalence

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

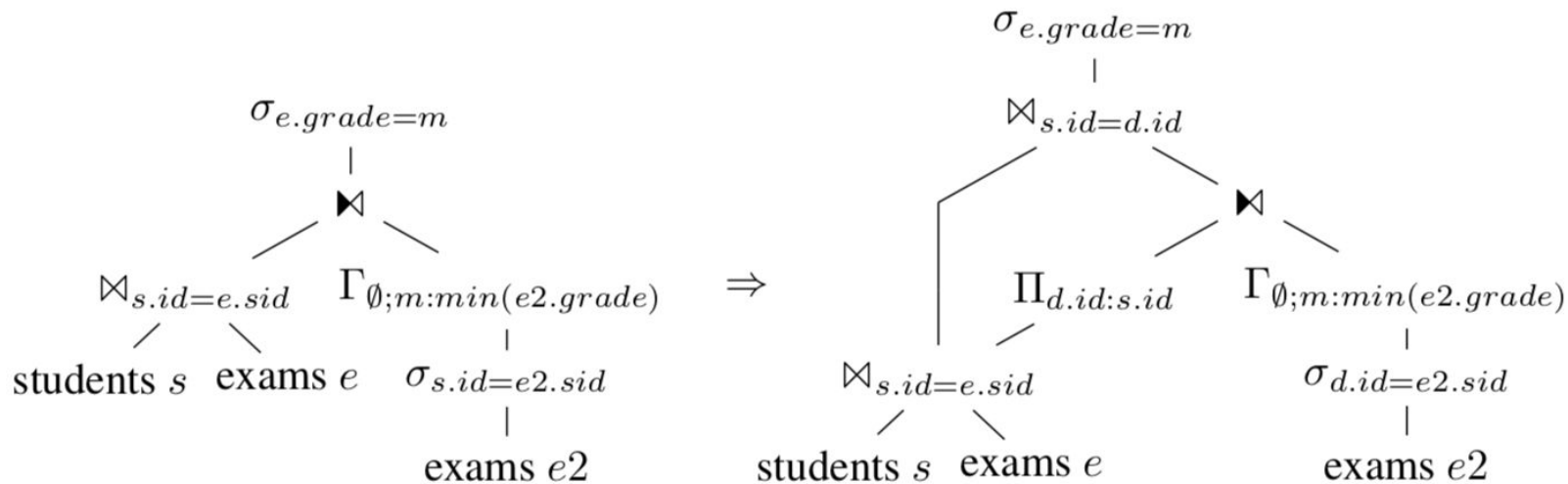
where  $D := \Pi_{\mathcal{F}(T_2) \cap \mathcal{A}(T_1)}(T_1)$ .

# Unnesting (Example)

```
Q1: select s.name, e.course
      from students s, exams e
      where s.id=e.sid and
            e.grade=(select min(e2.grade)
                      from exams e2
                      where s.id=e2.sid)
```



# Unnesting (Example)



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

where  $D := \Pi_{\mathcal{F}(T_2) \cap \mathcal{A}(T_1)}(T_1)$ .

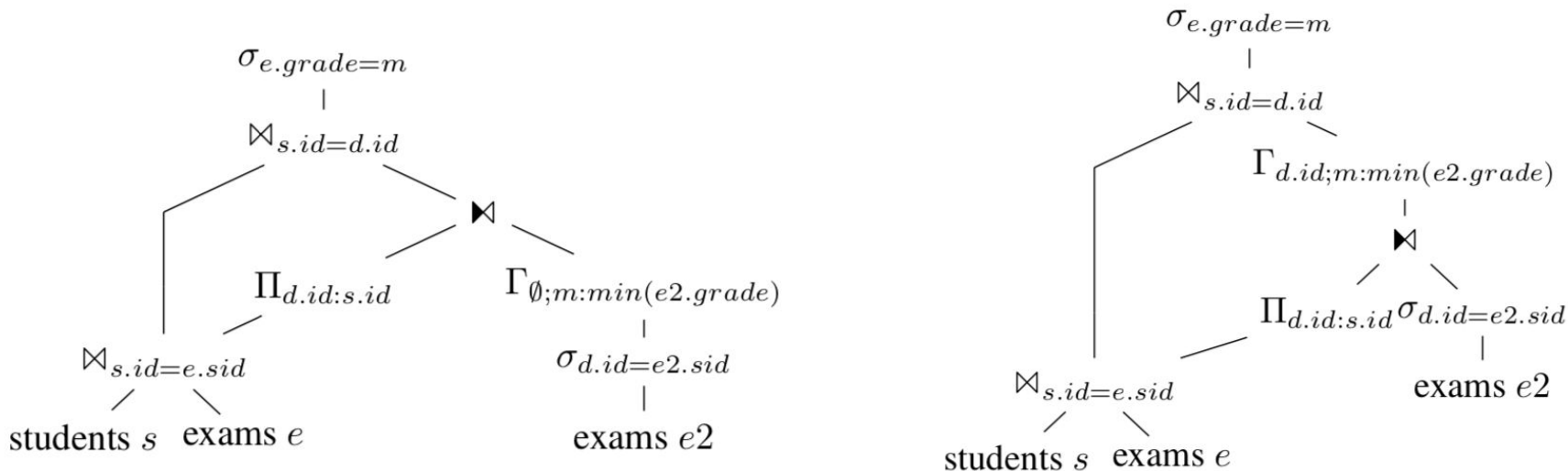


# Unnesting

- For group by operator

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

# Unnesting (Example)



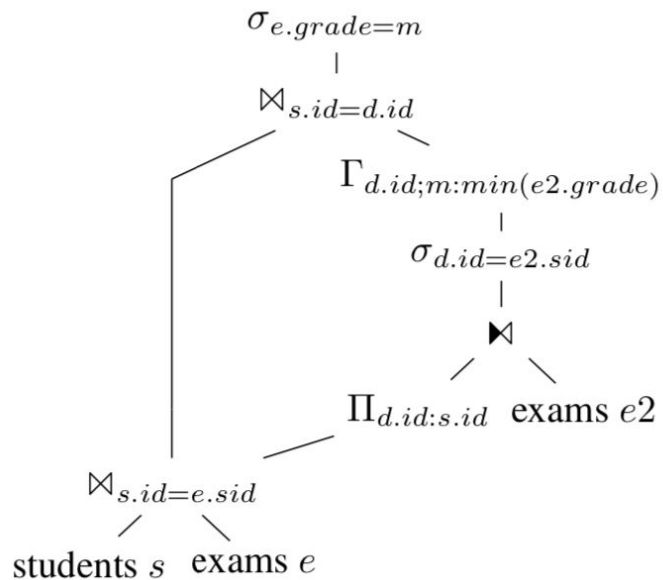
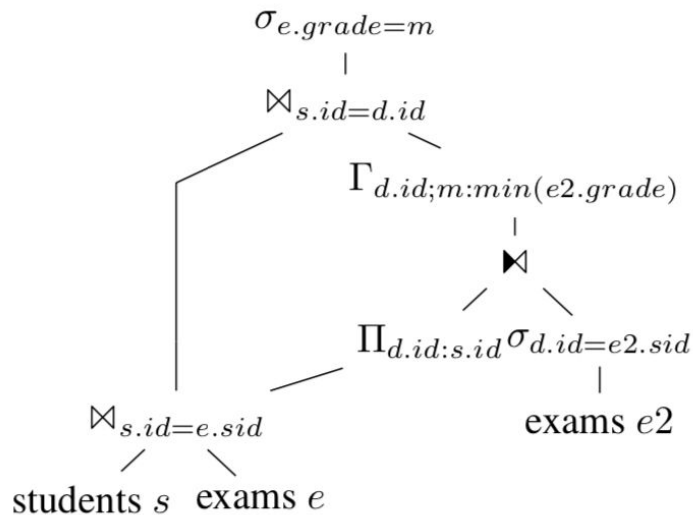
$$D \bowtie (\Gamma_{A;a:f}(T)) \equiv \Gamma_{A \cup \mathcal{A}(D);a:f}(D \bowtie T)$$

# Unnesting

- For selections, a push-down is very simple

$$D \bowtie \sigma_p(T_2) \equiv \sigma_p(D \bowtie T_2)$$

# Unnesting (Example)



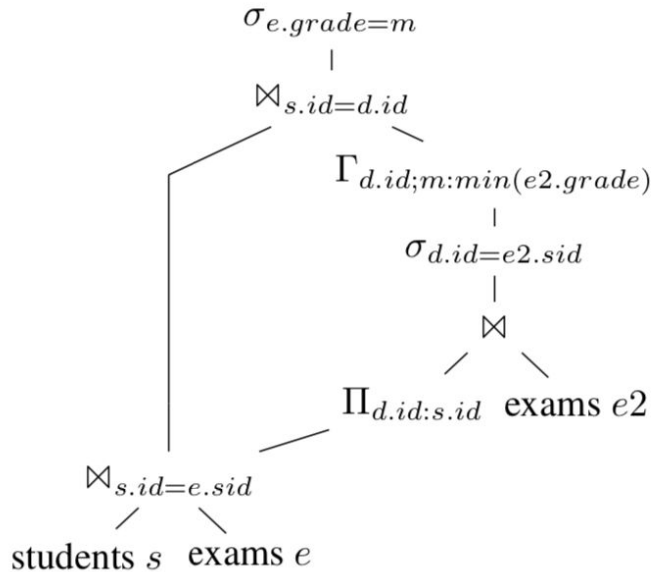
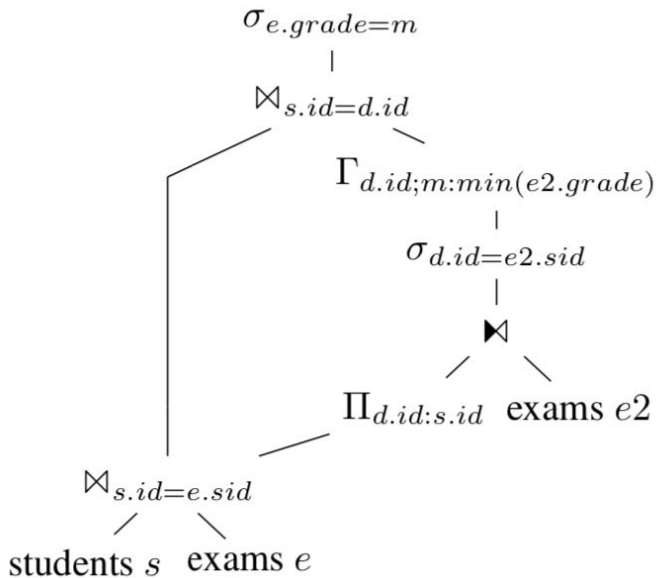
$$D \bowtie \sigma_p(T_2) \equiv \sigma_p(D \bowtie T_2)$$

# Unnesting

- The ultimate goal

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

## Unnesting (Example)



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

# Unnesting

- For projection

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

- For set operations

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

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

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

# Unnesting

- For another join, push-down is more complex

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



# Unnesting

- For outer join

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

$$D \bowtie (T_1 \bowtie_p T_2) \equiv (D \bowtie T_1) \bowtie_{p \wedge \text{natural join } D} (D \bowtie T_2).$$

# Unnesting

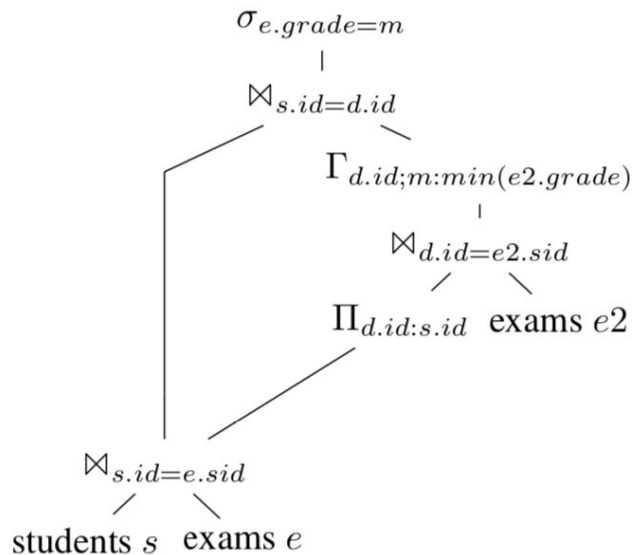
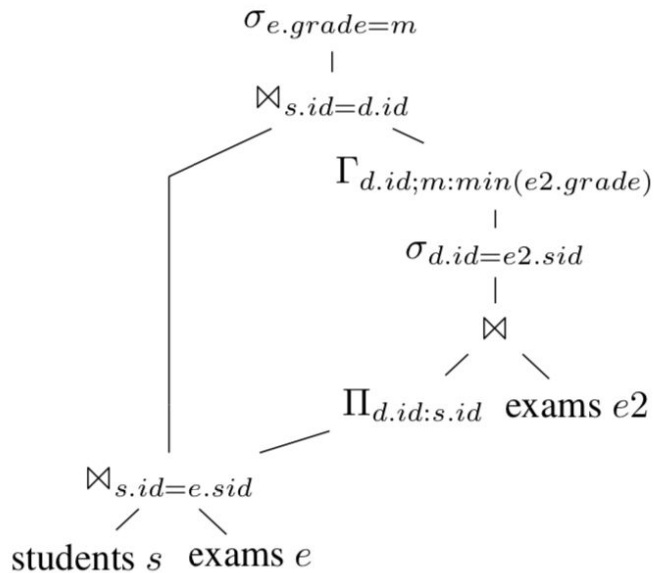
- For semi join and anti join

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

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

# Optimizations

- Final result ?

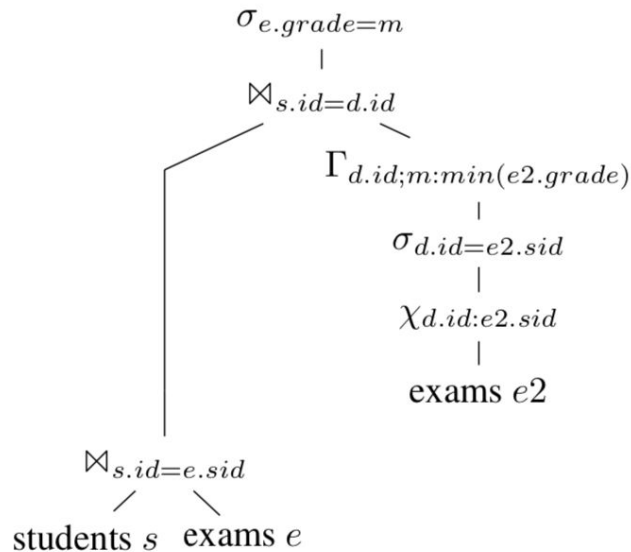
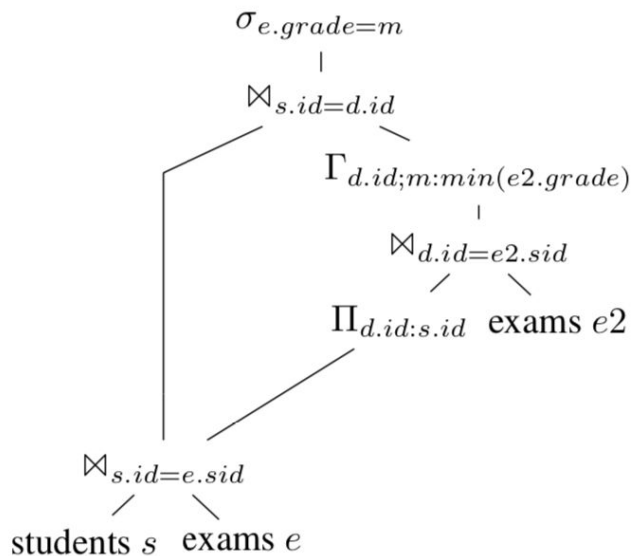


(pushing selections back down)

# Optimizations

- Eliminating  $D$  ( $D$  is a set)

$$D \bowtie T \subseteq \chi_{\mathcal{A}(D):B}(T) \text{ if } \exists B \subseteq \mathcal{A}(T) : \mathcal{A}(D) \equiv_C B$$



# Evaluation

- All experiments were run on an Intel i7-3930K with 64GB main memory.

	HyPer	HyPer Without Unnesting	PostgreSQL	PostgreSQL Without Unnesting
Query 1	<1ms	51ms	17ms	1,300ms
Query 2	42ms	408ms	N/A	12,099ms
TPCH-Q4	7ms	157,616ms		
TPCH-Q17	9ms	4,664ms		

# Thank You !

