Carnegie Mellon University

OPTIMIZE!

Database Query Optimization

Unnesting Queries

SPRING 2025 » SPECIAL TOPICS IN DATABASES » PROF. ANDY PAVLO

LAST CLASS

Parallelization of independent transformations in a top-down optimizer.

→ Another example of the need to track dependencies between parts of the query plan and optimization process.

This concludes the distinction between bottom-up and top-down methods.

SUBQUERIES

SQL allows a nested **SELECT** subquery to exist (almost?) anywhere in another query.

→ Projection, **FROM**, **WHERE**, **LIMIT**, **HAVING**

→ Results of the inner subquery are passed to the outer query.

Such nesting enables more expressive queries without having to use separate queries to prepare intermediate results.

Key Distinction: Uncorrelated vs. Correlated

UNCORRELATED SUBQUERY

An uncorrelated subquery does not reference any attributes from the (calling) outer query.

The DBMS only needs to logically execute the subquery once and reuse its result for all tuples in outer query.
→ Most DBMSs will do this.

```
SELECT name
  FROM students
 WHERE score =
  (SELECT MAX(score) FROM students);
```

CORRELATED SUBQUERY

A correlated subquery refers to one or more attributes from outside of the subquery (i.e., the outer query).

The DBMS logically evaluates the subquery on each tuple in the outer query because the result can change per tuple.

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

name	major	score
GZA	CompSci	90
RZA	CompSci	80
ODB	Streets	100

CORRELATED SUBQUERY

A correlated subquery refers to one or more attributes from outside of the subquery (i.e., the outer query).

The DBMS logically evaluates the subquery on each tuple in the outer query because the result can change per tuple.

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

name	major	score
GZA	CompSci	90
RZA	CompSci	80
ODB	Streets	100

s1.major='CompSci'

CORRELATED SUBQUERY

A correlated subquery refers to one or more attributes from outside of the subquery (i.e., the outer query).

The DBMS logically evaluates the subquery on each tuple in the outer query because the result can change per tuple.

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

name	major	score
GZA	CompSci	90
RZA	CompSci	80
ODB	Streets	100

s1.major='CompSci'

CORRELATED SUBQUERY

A correlated subquery refers to one or more attributes from outside of the subquery (i.e., the outer query).

The DBMS logically evaluates the subquery on each tuple in the outer query because the result can change per tuple.

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

name	major
GZA	CompSci

name	major	score
GZA	CompSci	90
RZA	CompSci	80
ODB	Streets	100

s1.major='CompSci'

MAX(s2.score)=90

CORRELATED SUBQUERY

A correlated subquery refers to one or more attributes from outside of the subquery (i.e., the outer query).

The DBMS logically evaluates the subquery on each tuple in the outer query because the result can change per tuple.

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

name	major
GZA	CompSci

name	major	score
GZA	CompSci	90
RZA	CompSci	80
ODB	Streets	100

s1.major='CompSci' MAX(s2.score)=90

s1.major='CompSci'

CORRELATED SUBQUERY

A correlated subquery refers to one or more attributes from outside of the subquery (i.e., the outer query).

The DBMS logically evaluates the subquery on each tuple in the outer query because the result can change per tuple.

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

name	major
GZA	CompSci

name	major	score
GZA	CompSci	90
RZA	CompSci	80
ODB	Streets	100

s1.major='CompSci' MAX(s2.score)=90

s1.major='CompSci'

CORRELATED SUBQUERY

A correlated subquery refers to one or more attributes from outside of the subquery (i.e., the outer query).

The DBMS logically evaluates the subquery on each tuple in the outer query because the result can change per tuple.

```sql
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

name	major
GZA	CompSci

name	major	score
GZA	CompSci	90
RZA	CompSci	80
ODB	Streets	100

s1.major='CompSci' MAX(s2.score)=90

s1.major='CompSci' MAX(s2.score)=90

CORRELATED SUBQUERY

A correlated subquery refers to one or more attributes from outside of the subquery (i.e., the outer query).

The DBMS logically evaluates the subquery on each tuple in the outer query because the result can change per tuple.

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

name	major
GZA	CompSci

name	major	score
GZA	CompSci	90
RZA	CompSci	80
ODB	Streets	100

s1.major='CompSci' MAX(s2.score)=90

s1.major='CompSci' MAX(s2.score)=90

s1.major='Streets'

CORRELATED SUBQUERY

A correlated subquery refers to one or more attributes from outside of the subquery (i.e., the outer query).

The DBMS logically evaluates the subquery on each tuple in the outer query because the result can change per tuple.

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

name	major
GZA	CompSci

name	major	score
GZA	CompSci	90
RZA	CompSci	80
ODB	Streets	100

s1.major='CompSci' MAX(s2.score)=90

s1.major='CompSci' MAX(s2.score)=90

s1.major='Streets'

CORRELATED SUBQUERY

A correlated subquery refers to one or more attributes from outside of the subquery (i.e., the outer query).

The DBMS logically evaluates the subquery on each tuple in the outer query because the result can change per tuple.

```sql
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

name	major
GZA	CompSci
ODB	Streets

name	major	score
GZA	CompSci	90
RZA	CompSci	80
ODB	Streets	100

s1.major='CompSci'	MAX(s2.score)=90
s1.major='CompSci'	MAX(s2.score)=90
s1.major='Streets'	**MAX(s2.score)=100**

CORRELATED SUBQUERY

The goal is for the optimizer to pull a correlated subquery up from an inner nesting level so that the DBMS can execute it as a join.

The optimizer needs to handle any amount of subquery nesting in any part of the query where it is allowed.

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

↓

```
SELECT s1.name, s1.major
  FROM students AS s1
  JOIN (SELECT major,
         MAX(score) AS max_score
         FROM students
         GROUP BY major) AS s2
    ON s1.major = s2.major
   AND s1.score = s2.max_score
```

TODAY'S AGENDA

Binding
Heuristic Rewriting
German-style Unnesting (2015)
German-style Unnesting (2025)

SUBQUERY BINDING

If you think of a subquery like a function call, then any column that can be passed to a function should be available to the subquery.

This can be challenging if the referenced columns are ambiguous.

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

```
SELECT name, major
  FROM students AS s1
 WHERE score = subquery(s1.major);
```

Source: Mark Raasveldt

SUBQUERY BINDING

SELECT:
→ Normal columns
→ **AGGREGATE/GROUP** columns

WHERE / GROUP BY:
→ Any normal column available

HAVING:
→ **AGGREGATE/GROUP** columns

ORDER BY:
→ Anything that can go in the root of **SELECT**.

LIMIT:
→ No correlated columns allowed.

```
SELECT (SELECT SUM(i1.i))
  FROM integers AS i1;
```

```
SELECT subquery(SUM(i1.i))
  FROM integers AS i1;
```

```
SELECT subquery(i1.i)
  FROM integers AS i1;
```

Source: Mark Raasveldt

SUBQUERY BINDING

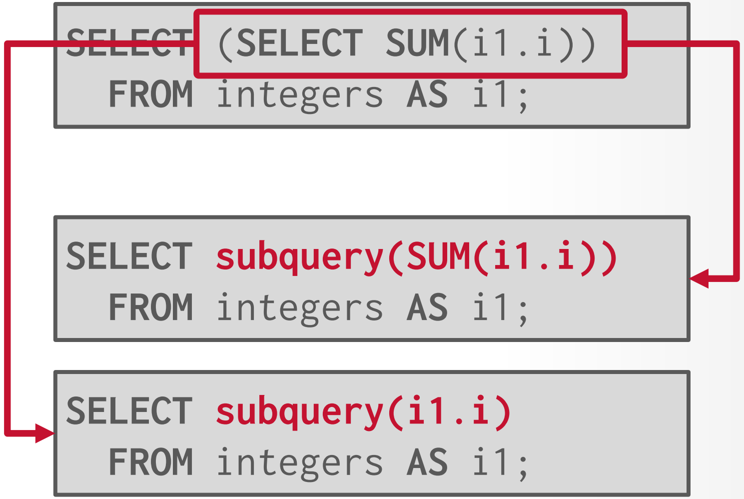
SELECT:
→ Normal columns
→ **AGGREGATE/GROUP** columns

WHERE / GROUP BY:
→ Any normal column available

HAVING:
→ **AGGREGATE/GROUP** columns

ORDER BY:
→ Anything that can go in the root of **SELECT**.

LIMIT:
→ No correlated columns allowed.

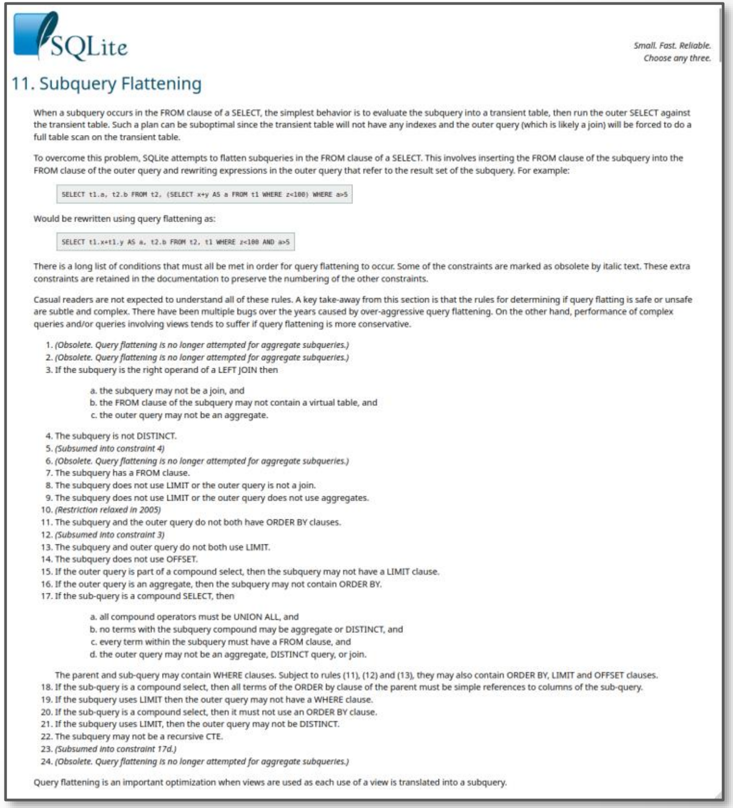


Source: Mark Raasveldt

HEURISTIC REWRITING

Since the early 1980s, optimizers relied on heuristics to identify specific query plan patterns to decorrelate nested subqueries.

The optimizer developer human codifies the patterns to look for when and how to decorrelate subqueries.



SQLite

Small. Fast. Reliable.
Choose any three.

11. Subquery Flattening

When a subquery occurs in the FROM clause of a SELECT, the simplest behavior is to evaluate the subquery into a transient table, then run the outer SELECT against the transient table. Such a plan can be suboptimal since the transient table will not have any indexes and the outer query (which is likely a join) will be forced to do a full table scan on the transient table.

To overcome this problem, SQLite attempts to flatten subqueries in the FROM clause of a SELECT. This involves inserting the FROM clause of the subquery into the FROM clause of the outer query and rewriting expressions in the outer query that refer to the result set of the subquery. For example:

```
SELECT t1.a, t2.b FROM t2, (SELECT x+y AS a FROM t1 WHERE z<100) WHERE a>5
```

Would be rewritten using query flattening as:

```
SELECT t1.x+t1.y AS a, t2.b FROM t2, t1 WHERE z<100 AND a>5
```

There is a long list of conditions that must all be met in order for query flattening to occur. Some of the constraints are marked as obsolete by italic text. These extra constraints are retained in the documentation to preserve the numbering of the other constraints.

Casual readers are not expected to understand all of these rules. A key take-away from this section is that the rules for determining if query flatting is safe or unsafe are subtle and complex. There have been multiple bugs over the years caused by over-aggressive query flattening. On the other hand, performance of complex queries and/or queries involving views tends to suffer if query flattening is more conservative.

1. *(Obsolete. Query flattening is no longer attempted for aggregate subqueries.)*
2. *(Obsolete. Query flattening is no longer attempted for aggregate subqueries.)*
3. If the subquery is the right operand of a LEFT JOIN then
   a. the subquery may not be a join, and
   b. the FROM clause of the subquery may not contain a virtual table, and
   c. the outer query may not be an aggregate.
4. The subquery is not DISTINCT.
5. *(Subsumed into constraint 4)*
6. *(Obsolete. Query flattening is no longer attempted for aggregate subqueries.)*
7. The subquery has a FROM clause.
8. The subquery does not use LIMIT or the outer query is not a join.
9. The subquery does not use LIMIT or the outer query does not use aggregates.
10. *(Restriction relaxed in 2005)*
11. The subquery and the outer query do not both have ORDER BY clauses.
12. *(Subsumed into constraint 3)*
13. The subquery and outer query do not both use LIMIT.
14. The subquery does not use OFFSET.
15. If the outer query is part of a compound select, then the subquery may not have a LIMIT clause.
16. If the outer query is an aggregate, then the subquery may not contain ORDER BY.
17. If the sub-query is a compound SELECT, then
    a. all compound operators must be UNION ALL, and
    b. no terms with the subquery compound may be aggregate or DISTINCT, and
    c. every term within the subquery must have a FROM clause, and
    d. the outer query may not be an aggregate, DISTINCT query, or join.

    The parent and sub-query may contain WHERE clauses. Subject to rules (11), (12) and (13), they may also contain ORDER BY, LIMIT and OFFSET clauses.
18. If the sub-query is a compound select, then all terms of the ORDER by clause of the parent must be simple references to columns of the sub-query.
19. If the subquery uses LIMIT then the outer query may not have a WHERE clause.
20. If the sub-query is a compound select, then it must not use an ORDER BY clause.
21. If the subquery uses LIMIT, then the outer query may not be DISTINCT.
22. The subquery may not be a recursive CTE.
23. *(Subsumed into constraint 17d.)*
24. *(Obsolete. Query flattening is no longer attempted for aggregate subqueries.)*

Query flattening is an important optimization when views are used as each use of a view is translated into a subquery.

ON OPTIMIZING AN SQL-LIKE NESTED QUERY
ACM TDS 1982

MAGIC SETS

Early technique for rewriting queries to include auxiliary "magic" tables that act as filters to reduce the amount of data processed during query execution.

Move correlated subqueries out of **WHERE** clause and into **FROM** clause.

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

```
SELECT s1.name, s1.major
  FROM students AS s1
  JOIN (SELECT major,
         MAX(score) AS max_score
         FROM students
         GROUP BY major) AS magic
    ON s1.major = magic.major
   AND s1.score = magic.max_score
```

COMPLEX QUERY DECORRELATION
ICDE 1996

MSSQL HEURISTICS

Use a set of small, independent, and orthogonal optimizations that collectively remove correlated subqueries.

Remove correlations by rewriting **APPLY** operators into standard relational algebra operators like outer joins.

$$
\begin{aligned}
R\ \mathcal{A}^{\otimes}\ E &= R \otimes_{\text{true}} E, & (1)\\
&\text{if no parameters in } E \text{ resolved from } R & \\
R\ \mathcal{A}^{\otimes}\ (\sigma_p E) &= R \otimes_p E, & (2)\\
&\text{if no parameters in } E \text{ resolved from } R & \\
R\ \mathcal{A}^{\times}\ (\sigma_p E) &= \sigma_p(R\ \mathcal{A}^{\times}\ E) & (3)\\
R\ \mathcal{A}^{\times}\ (\pi_v E) &= \pi_{v \cup \text{columns}(R)}(R\ \mathcal{A}^{\times}\ E) & (4)\\
R\ \mathcal{A}^{\times}\ (E_1 \cup E_2) &= (R\ \mathcal{A}^{\times}\ E_1) \cup (R\ \mathcal{A}^{\times}\ E_2) & (5)\\
R\ \mathcal{A}^{\times}\ (E_1 - E_2) &= (R\ \mathcal{A}^{\times}\ E_1) - (R\ \mathcal{A}^{\times}\ E_2) & (6)\\
R\ \mathcal{A}^{\times}\ (E_1 \times E_2) &= (R\ \mathcal{A}^{\times}\ E_1) \bowtie_{R.key} (R\ \mathcal{A}^{\times}\ E_2) & (7)\\
R\ \mathcal{A}^{\times}\ (\mathcal{G}_{A,F} E) &= \mathcal{G}_{A \cup \text{columns}(R),F}(R\ \mathcal{A}^{\times}\ E) & (8)\\
R\ \mathcal{A}^{\times}\ (\mathcal{G}^{1}_{F} E) &= \mathcal{G}_{\text{columns}(R),F'}(R\ \mathcal{A}^{\text{LOJ}}\ E) & (9)
\end{aligned}
$$

ORTHOGONAL OPTIMIZATION OF SUBQUERIES AND AGGREGATION
SIGMOD 2001

HEURISTIC REWRITING

Advantages:

→ Transformed queries are more efficient.
→ Decision to decorrelate can be a cost-based decision.
→ Easy to control decorrelation by enabling/disabling rules.

Disadvantages:

→ Hard to write rules for all possible correlations scenarios.
→ Changing a small part of a query can make rules ineffective
→ Maintaining transformation rules is a difficult.
→ Handling all edge cases is exceedingly difficult.

Source: Mayank Baranwal

GERMAN-STYLE UNNESTING (2015)

Bottom-up method to eliminate dependent joins one-at-a-time by manipulating the query plan at the algebra level until the join's RHS no longer depends on the LHS.

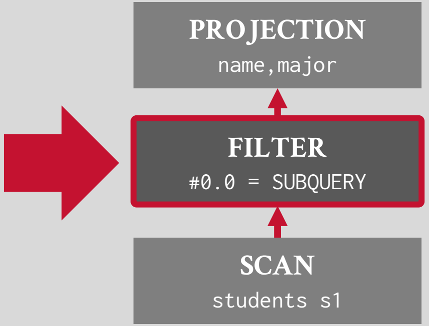
The optimizer then converts dependent joins to regular joins.

→ Some queries switch from a **$O(n^2)$** nested-loop join to a **$O(n)$** hash join.

UNNESTING ARBITRARY QUERIES
BTW 2015

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```



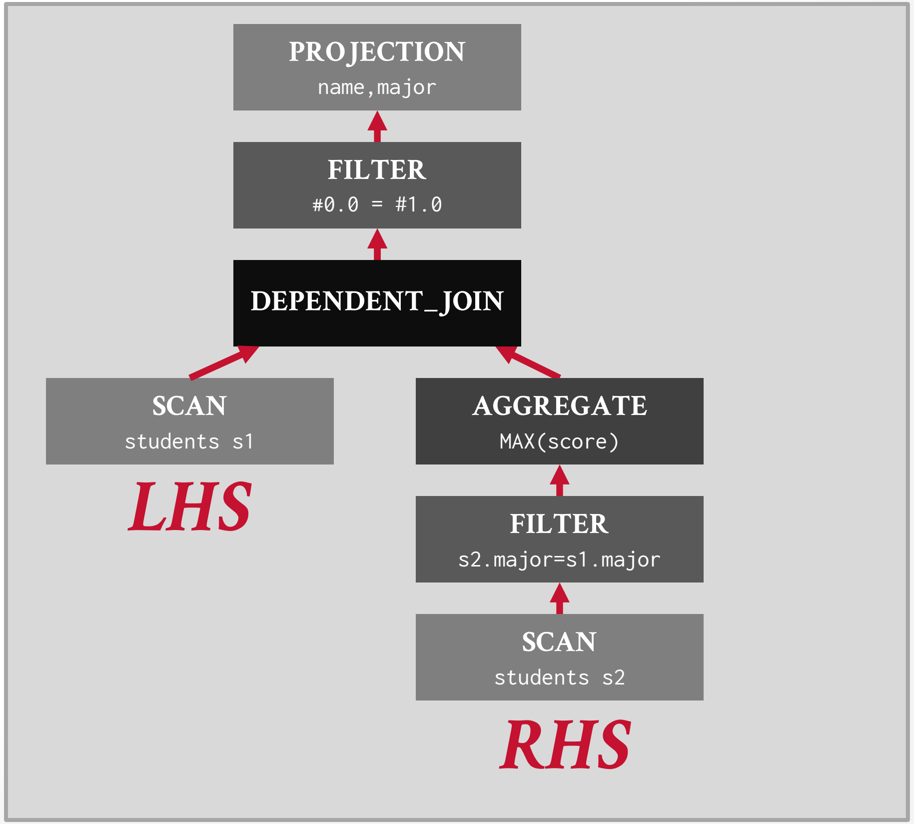
Introduce a **dependent join** logical operator to execute RHS once for every tuple in LHS.

Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Introduce a **dependent join** logical operator to execute RHS once for every tuple in LHS.

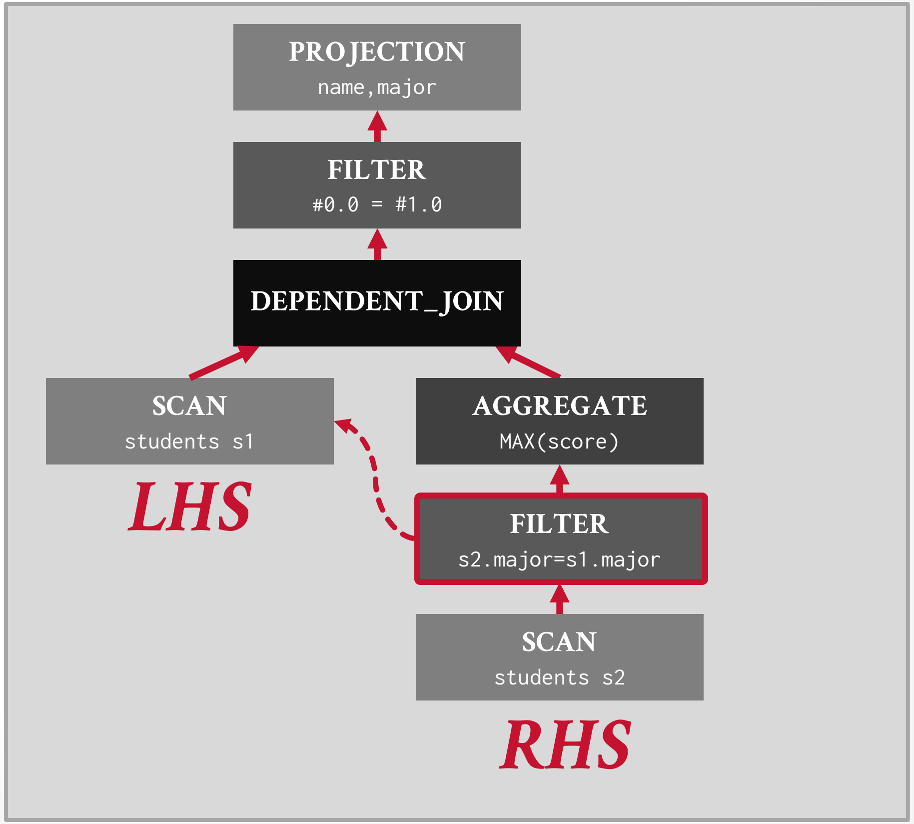


Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Introduce a **dependent join** logical operator to execute RHS once for every tuple in LHS.



Source: Mark Raasveldt

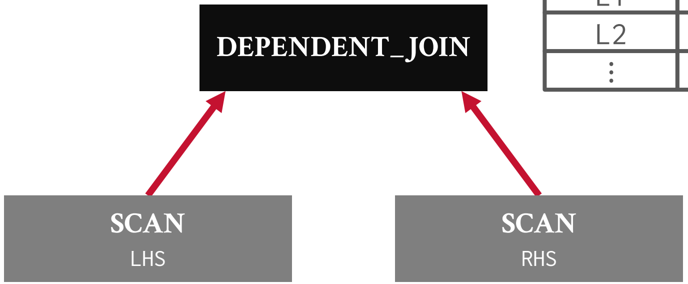
DEPENDENT JOIN

New **dependent join** relational algebra operator that denotes a correlated subquery.

→ Evaluate RHS of the join for every tuple on the LHS.

→ The operator combine results from every execution and return them as its output.

id	id
L1	R1
L1	R2
L2	R2
⋮	⋮



id
L1
L2
L3

id
R1
R2
R3

Source: Mayank Baranwal

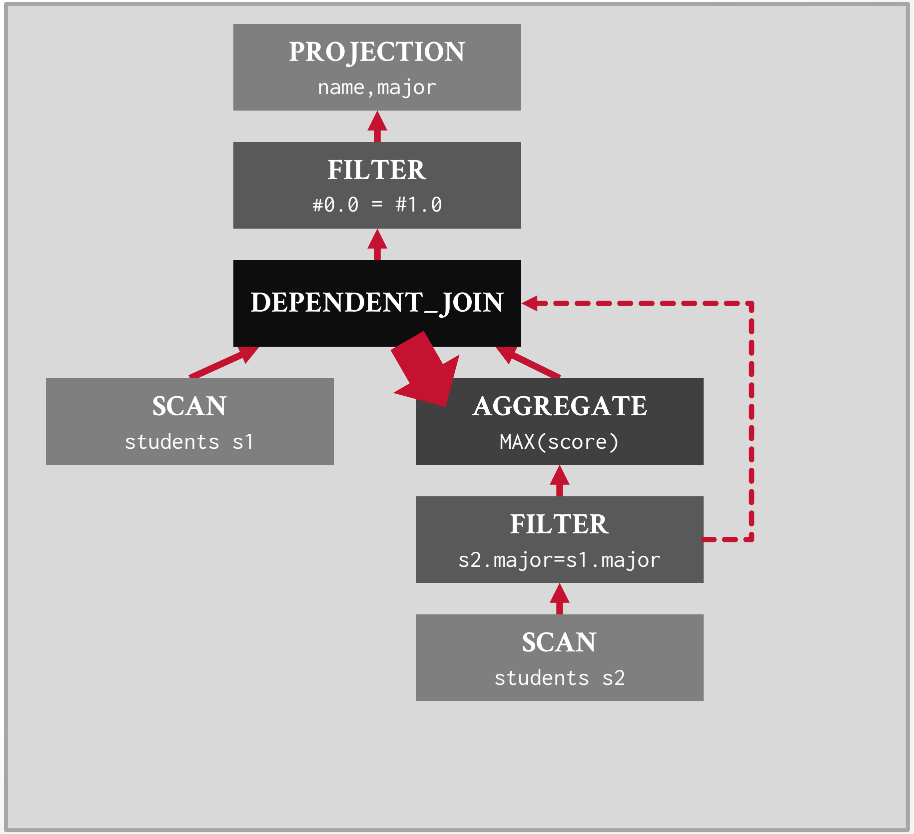
FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Push dependent join down into the RHS of the plan.

Only need to execute RHS once for every unique combination of correlated columns.

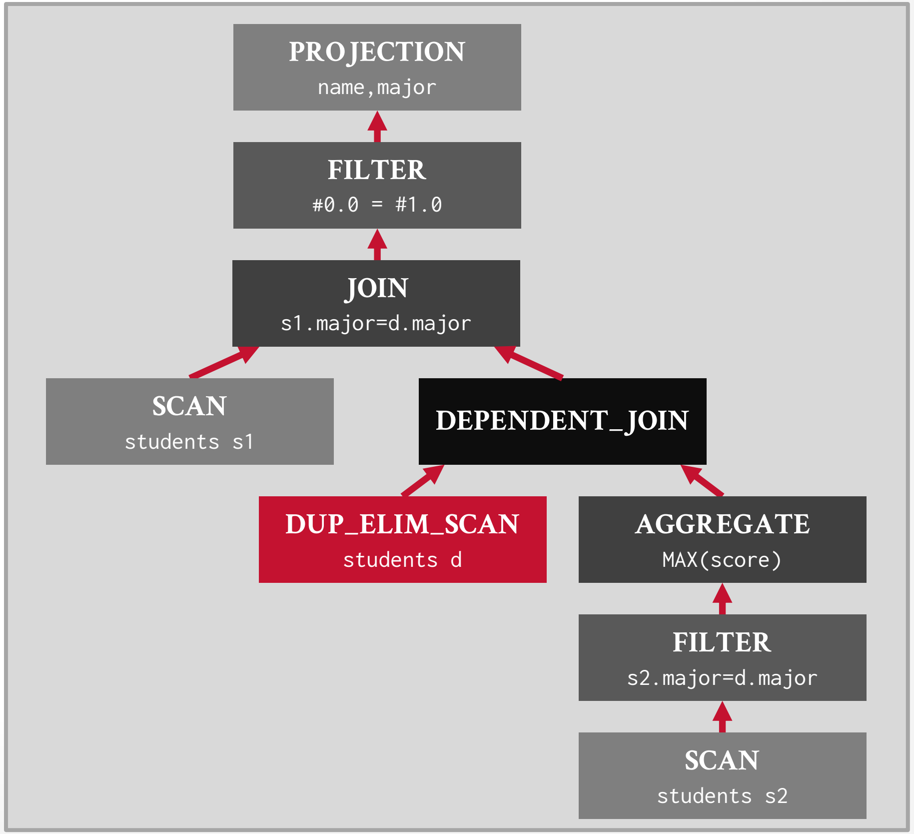
→ Duplicate Elimination Scan



Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```



Push dependent join down into the RHS of the plan.

Only need to execute RHS once for every unique combination of correlated columns.

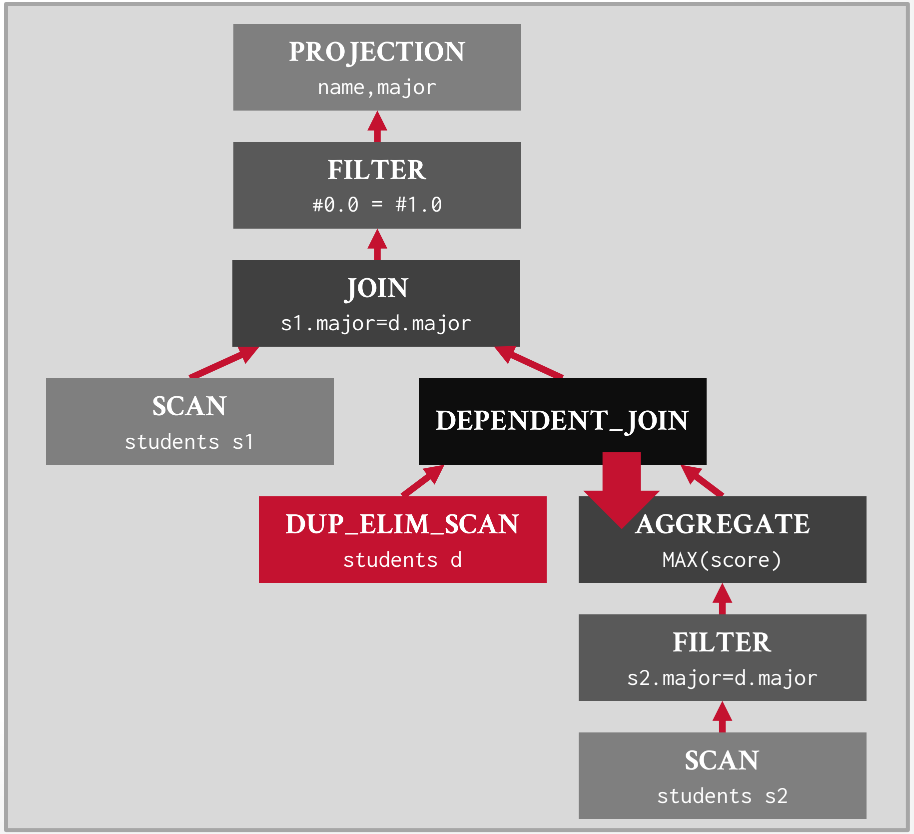
→ Duplicate Elimination Scan

Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Keeping pushing dependent join as far down into the plan as is possible.

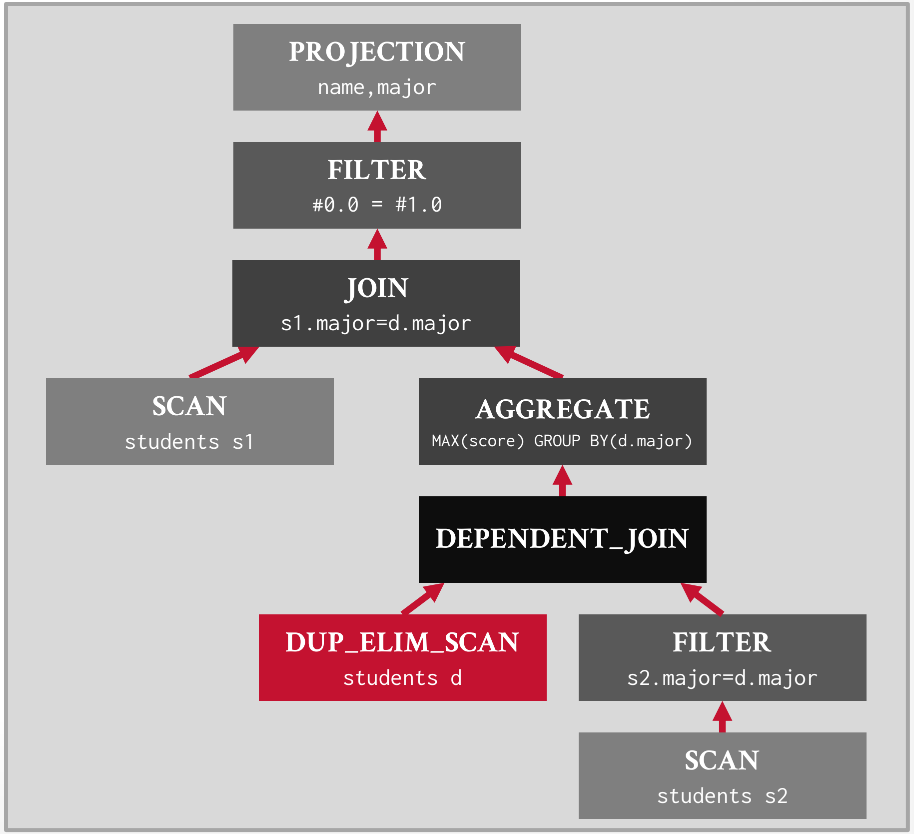


Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Keeping pushing dependent join as far down into the plan as is possible.

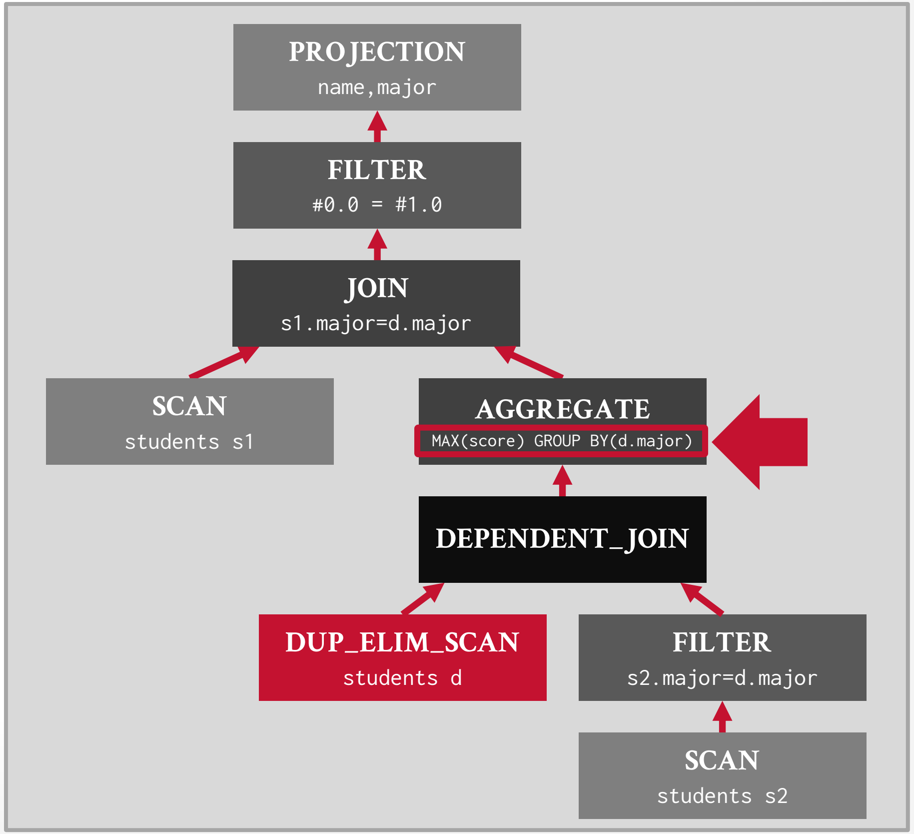


Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Keeping pushing dependent join as far down into the plan as is possible.

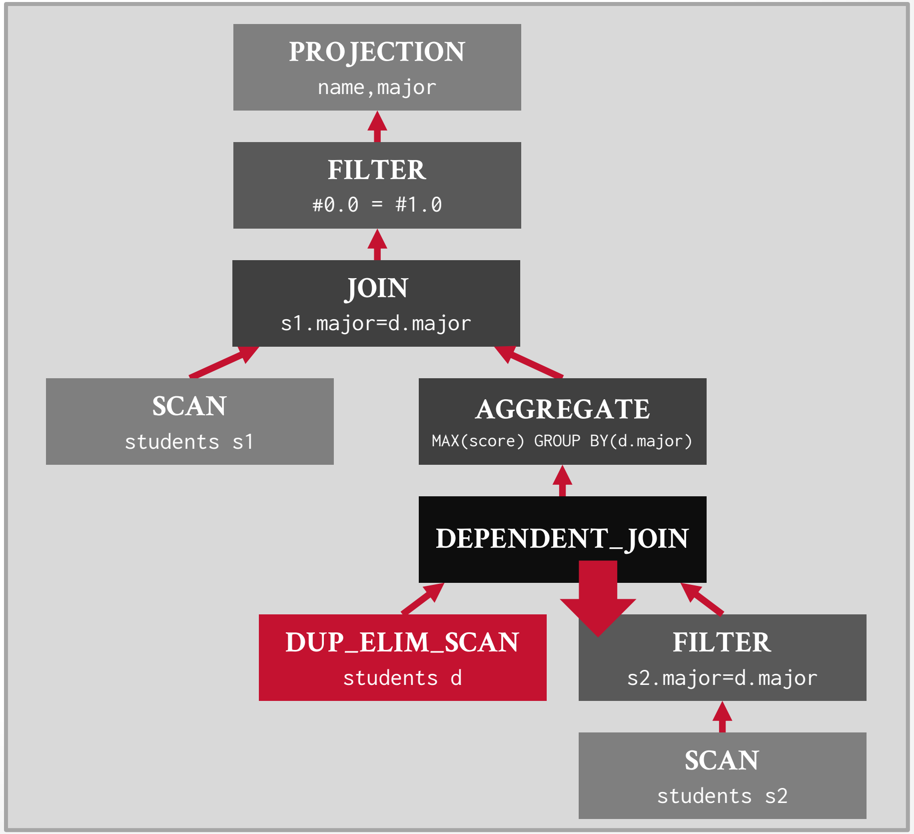


Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Keeping pushing dependent join as far down into the plan as is possible.

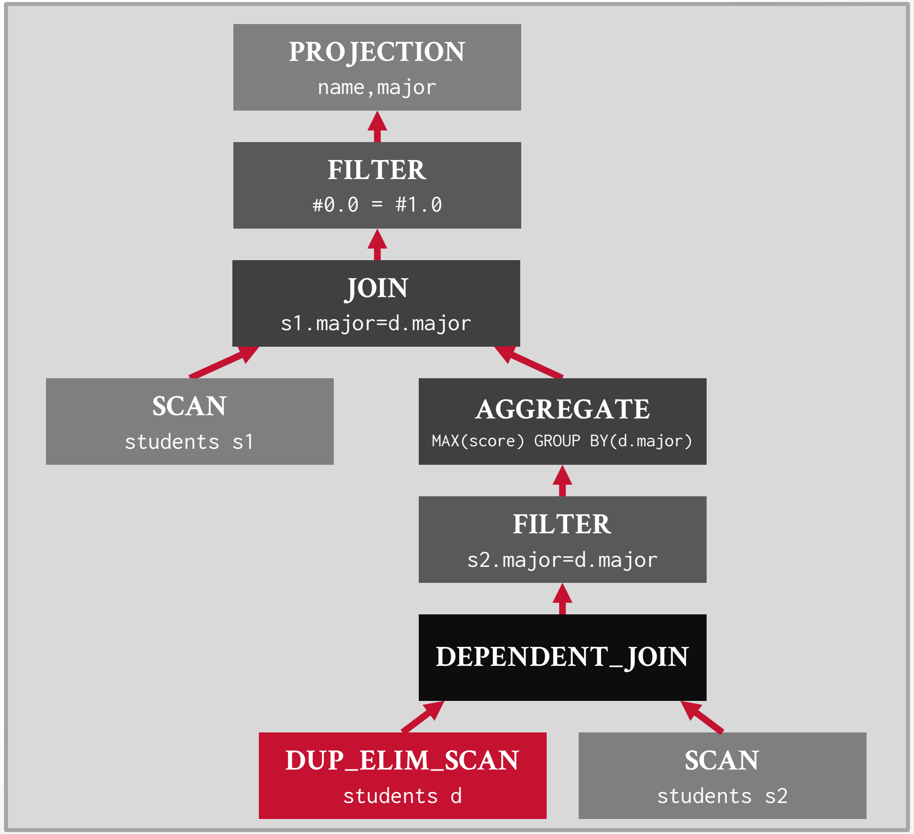


Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
      (SELECT MAX(s2.score)
         FROM students AS s2
        WHERE s2.major = s1.major);
```

Keeping pushing dependent join as far down into the plan as is possible.

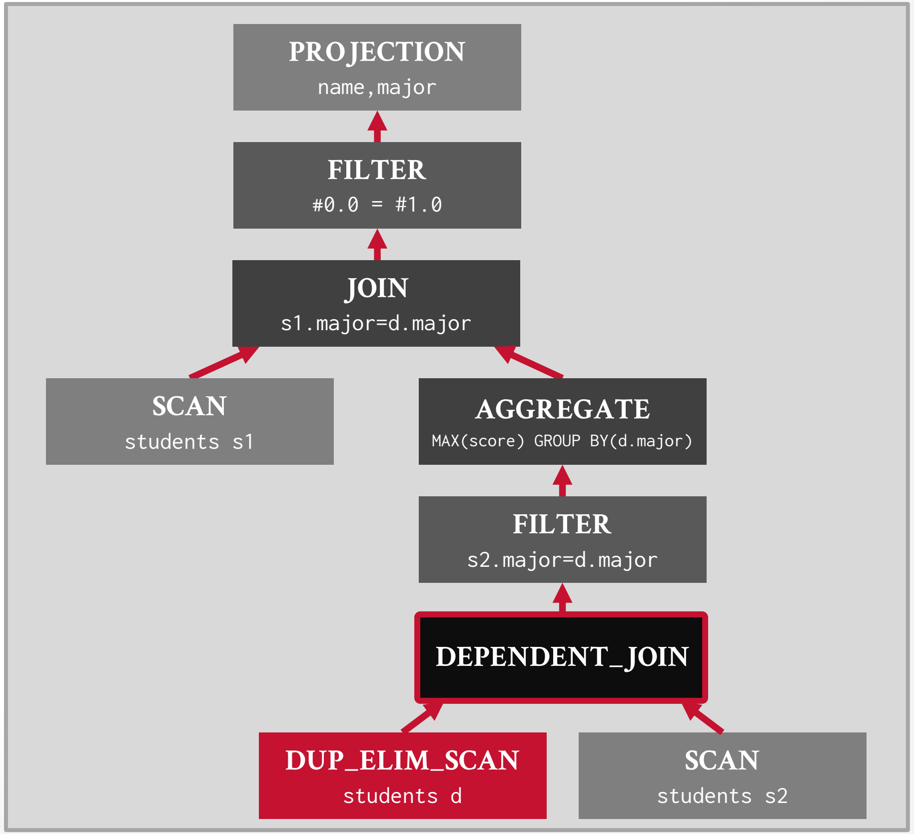


Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Convert the **dependent join** operator into a **cross join**.

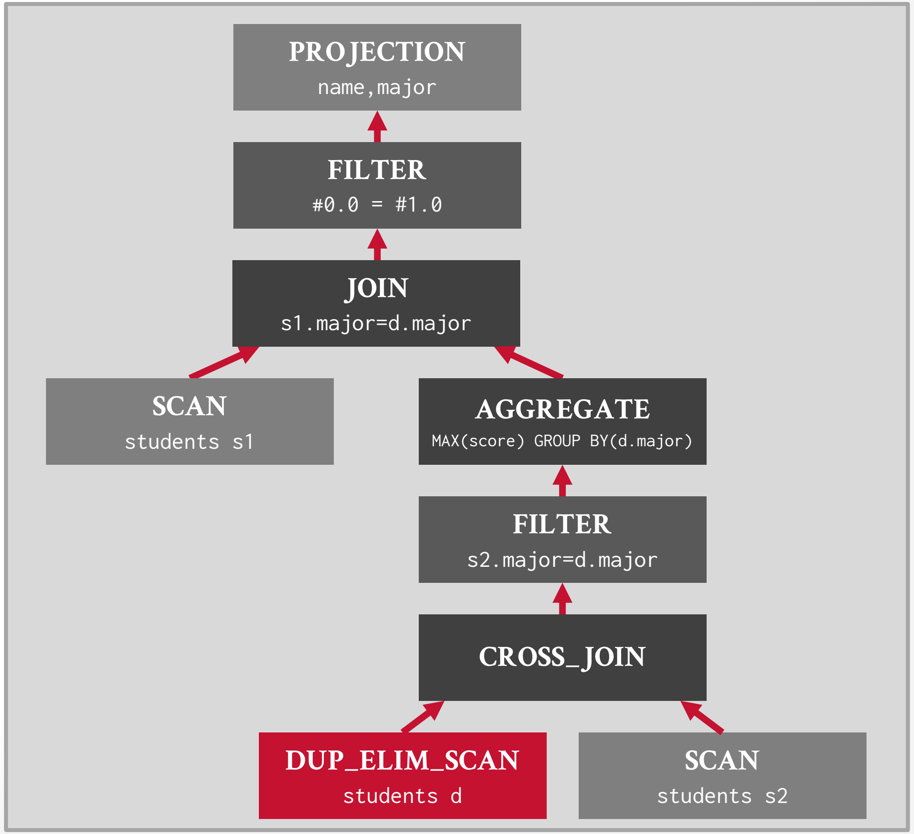


Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Convert the **dependent join** operator into a **cross join**.



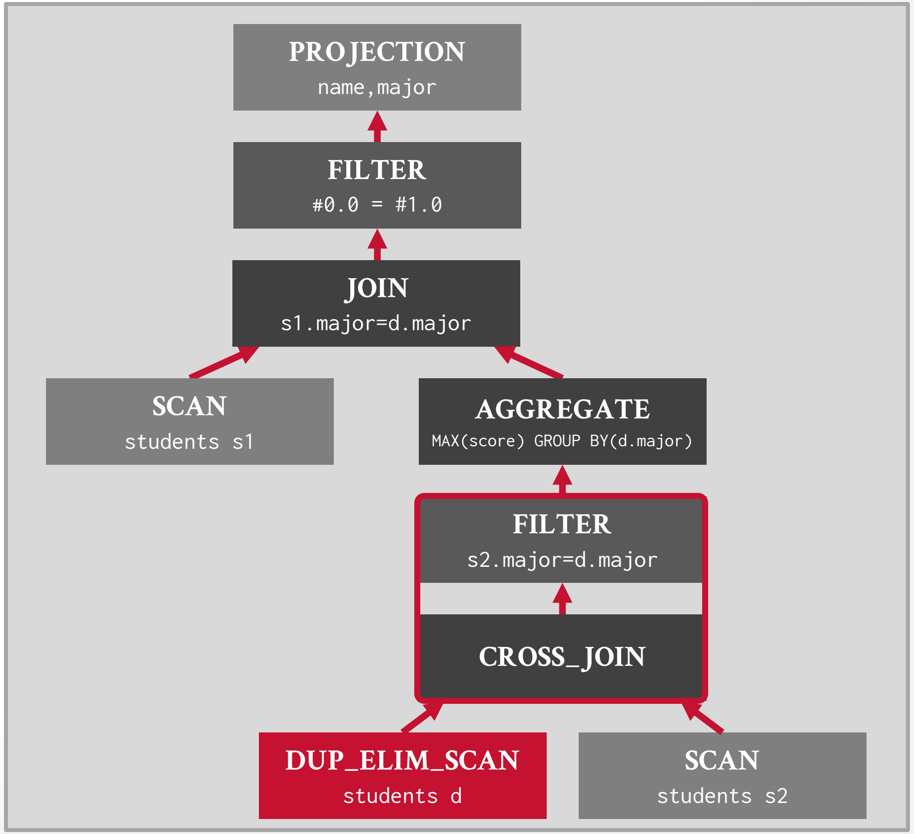
Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```sql
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Convert the **dependent join** operator into a **cross join**.

Then convert the **cross join** into an **inner join**.



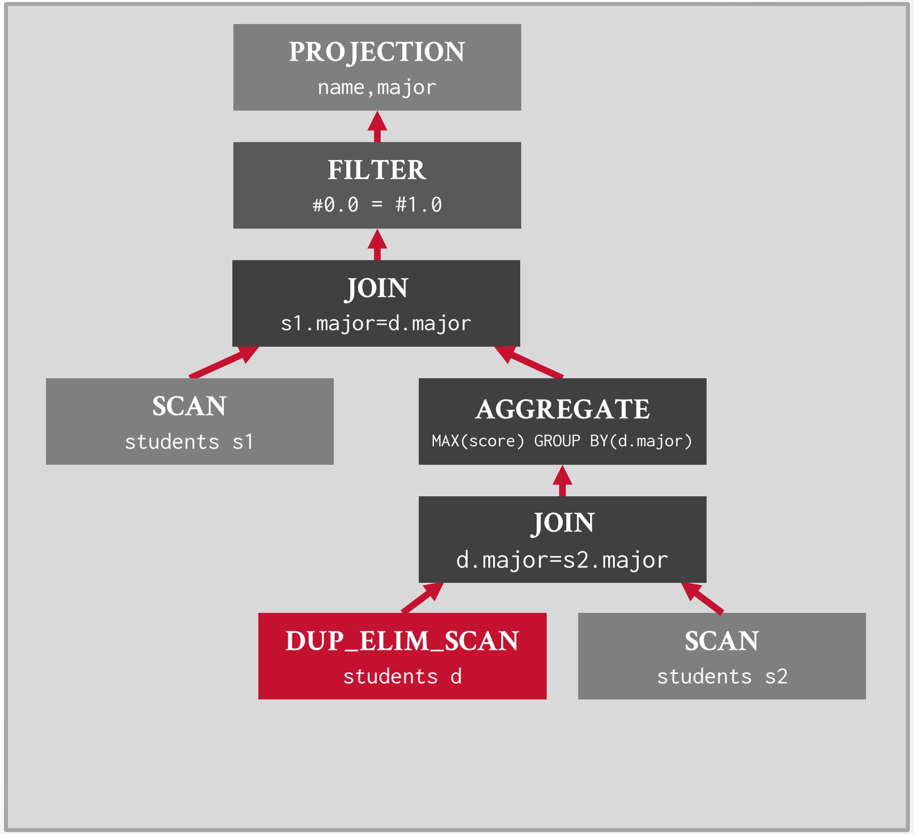
Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
      (SELECT MAX(s2.score)
         FROM students AS s2
        WHERE s2.major = s1.major);
```

Convert the **dependent join** operator into a **cross join**.

Then convert the **cross join** into an **inner join**.



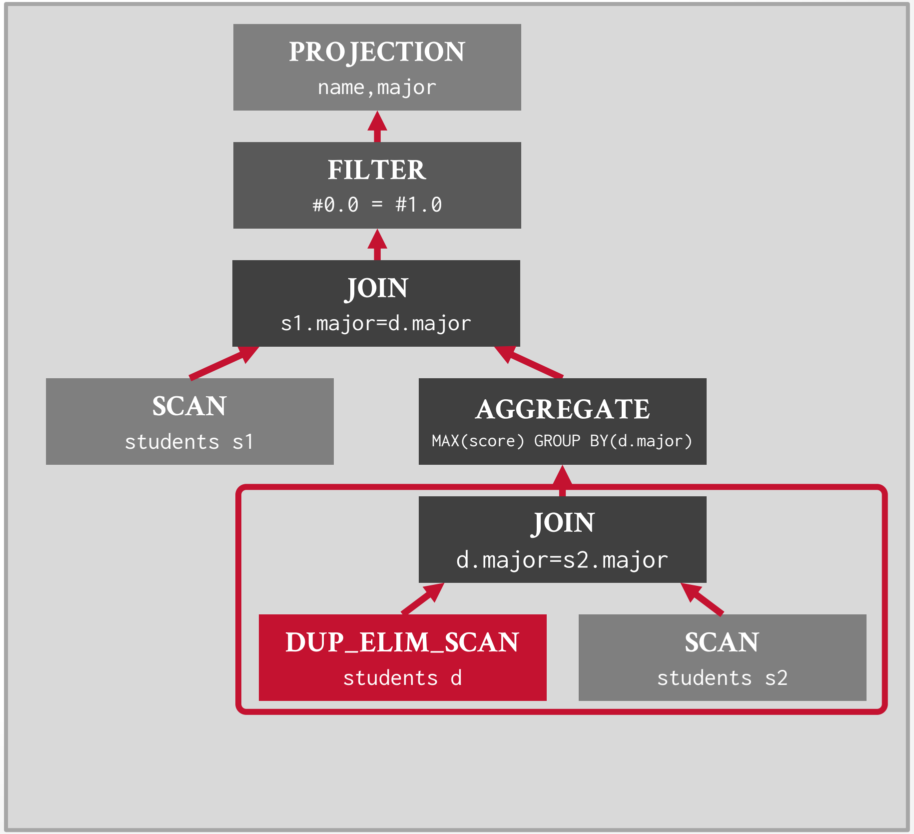
Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Remove duplicate elimination scan entirely.

Remove the filter above the new join.



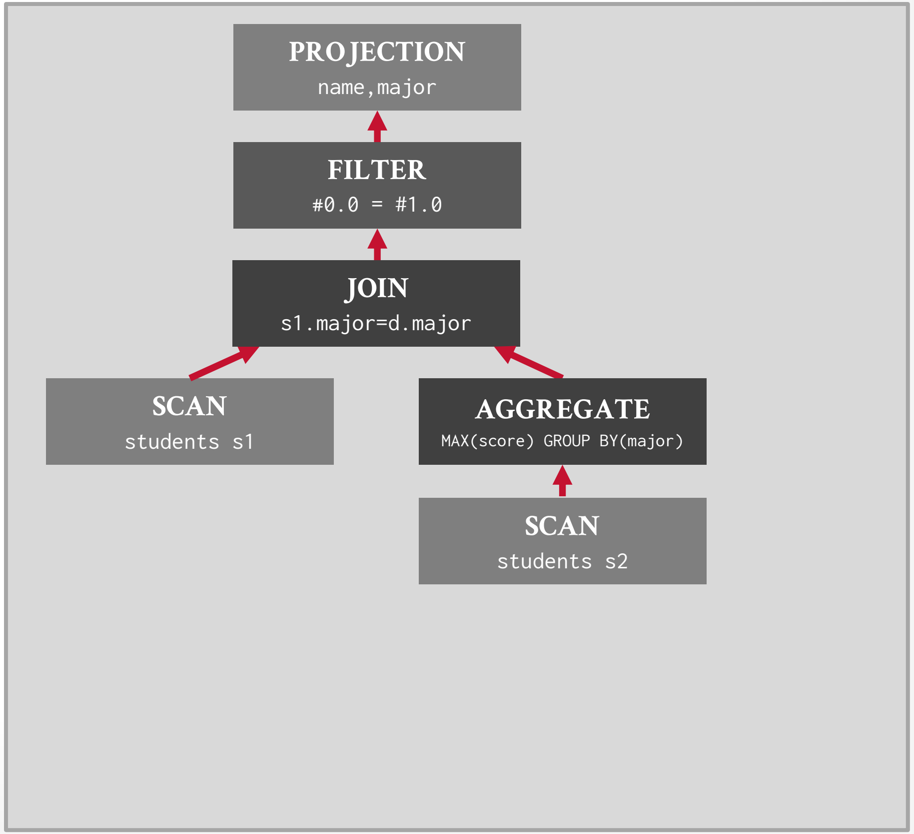
Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Remove duplicate elimination scan entirely.

Remove the filter above the new join.



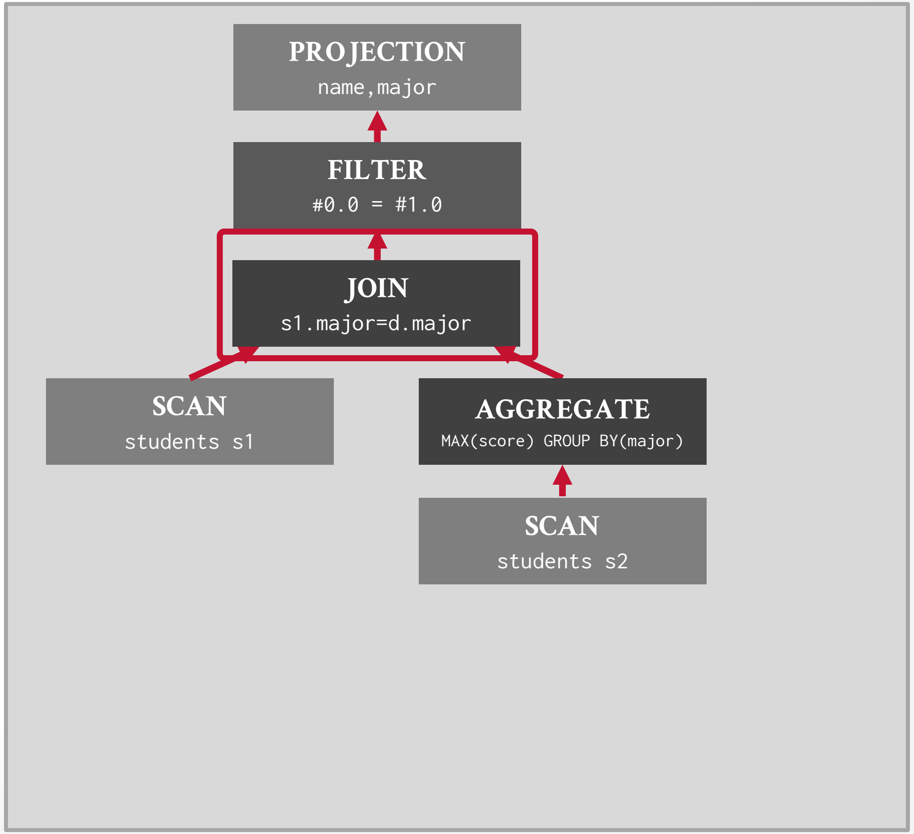
Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Remove duplicate elimination scan entirely.

Remove the filter above the new join.



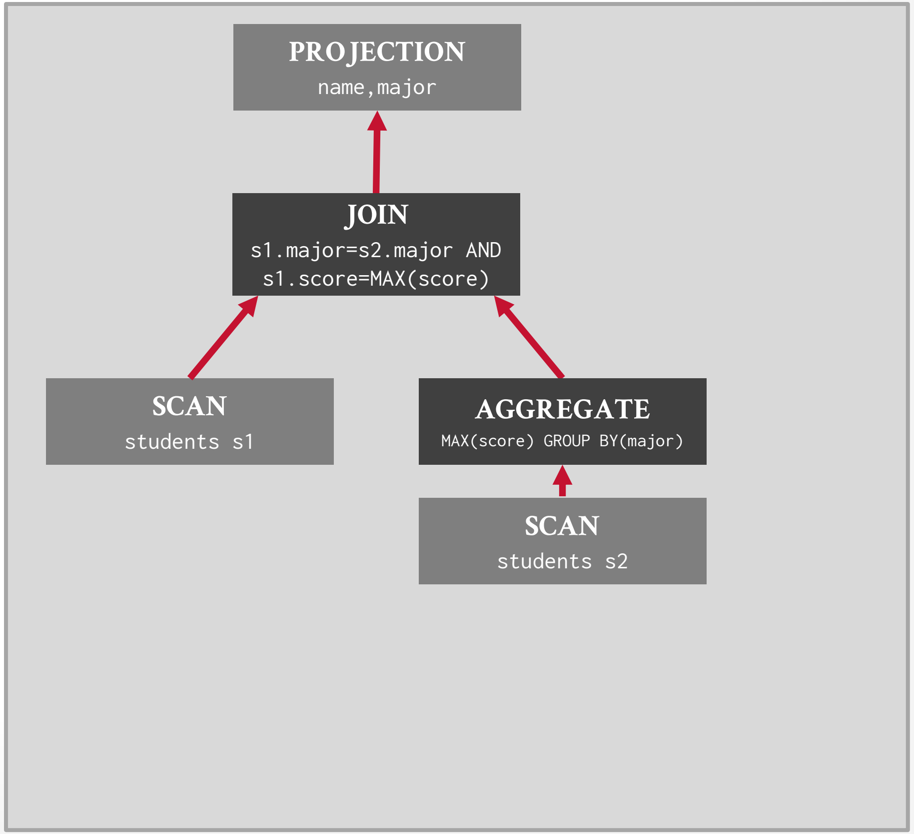
Source: Mark Raasveldt

FLATTENING CORRELATED QUERIES

```
SELECT name, major
  FROM students AS s1
 WHERE score =
       (SELECT MAX(s2.score)
          FROM students AS s2
         WHERE s2.major = s1.major);
```

Remove duplicate elimination scan entirely.

Remove the filter above the new join.



Source: Mark Raasveldt

OBSERVATION

The 2015 unnesting approach handles most queries.

→ Known implementations in HyPer, Umbra, DuckDB, and DataBricks (partial).

But for queries with multiple nested dependent subqueries where rewriting to remove each dependent join one at a time leads to inefficient query plans.

OBSERVATION

The 2015 … handles most queries.

→ Known … DataBr…

2.3 Limitations of the Bottom-Up Approach

While the bottom-up approach handles most queries just fine, it unfortunately degenerates in some corner cases. We were originally notified about this by Sam Arch, who translated complex UDFs into pure SQL [Fr24]. There, similar to our original example in Figure 1, it could happen that dependent subqueries are nested inside each other. We show a variation[2] below.

But for q… subqueries where rewriting to remove … … time leads to inefficient …

Similarly, for the original query from Sam Arch that motivated this work, which is procbench UDF Query 18 [GR21] after passing through Apfel [FHG22]: The unnesting strategy from [NK15] leads to memory exhaustion, while with our new top-down unnesting Umbra answers the query in 251ms on TPC-DS SF1.

OBSERVATION

The 2015 ... handles most queries.

→ Known ... DataBr...

...ere rewriting to remove ...

... at a time leads to inefficient

2.3 Limitations of the Bottom-Up Approach

While the bottom-up approach handles most queries just fine, it unfortunately degenerates in some corner cases. We were originally notified about this by Sam Arch, who translated ...UDFs into pure SQL [Fr24]. There, similar to our original example in Figure 1, it ...pen that dependent subqueries are nested inside each other. We show a variation[2]

...iginal query from Sam Arch that motivated this work, which is procbench ...R21] after passing through Apfel [FHG22]: The unnesting strategy from ...emory exhaustion, while with our new top-down unnesting Umbra ...n 251ms on TPC-DS SF1.

```sql
SELECT
  (SELECT "ifresult4".*
   FROM LATERAL
     (SELECT NULL :: numeric AS "numweb") AS "let0"("numweb"), LATERAL
     (SELECT NULL :: numeric AS "numstore") AS "let1"("numstore"), LATERAL
     (SELECT NULL :: numeric AS "numcat") AS "let2"("numcat"), LATERAL
     (SELECT TRUE AS "q4_1") AS "let3"("q4_1"), LATERAL (
                                     (SELECT "ifresult7".*
                                      FROM LATERAL
                                        (SELECT
                                           (SELECT sum("web_sales"."ws_net_paid_inc_ship_tax") AS "sum"
                                            FROM web_sales AS "web_sales"
                                            WHERE "web_sales"."ws_bill_customer_sk" = "c_customer_sk") AS "numweb_6") AS "let5"("numweb_6"), LATERAL
                                        (SELECT TRUE AS "q8_2") AS "let6"("q8_2"), LATERAL (
                                                    (SELECT "ifresult10".*
                                                     FROM LATERAL
                                                       (SELECT
                                                          (SELECT sum("store_sales"."ss_net_paid_inc_tax") AS "sum"
                                                           FROM store_sales AS "store_sales"
                                                           WHERE "store_sales"."ss_customer_sk" = "c_customer_sk") AS "numstore_5") AS "let8"("numstore_5"), LATERAL
                                                       (SELECT TRUE AS "q12_3") AS "let9"("q12_3"), LATERAL (
                                                                 (SELECT ("numweb_6" + "numstore_5") + "numcat_4" AS "result"
                                                                  FROM LATERAL
                                                                    (SELECT
                                                                       (SELECT sum("catalog_sales"."cs_net_paid_inc_ship_tax") AS "sum"
                                                                        FROM catalog_sales AS "catalog_sales"
                                                                        WHERE "catalog_sales"."cs_bill_customer_sk" = "c_customer_sk") AS "numcat_4") AS "let11"("numcat_4")
                                                                  WHERE NOT "q12_3" IS DISTINCT
                                                                    FROM TRUE)
                                                                 UNION ALL
                                                                  (SELECT ("numweb_6" + "numstore_5") + "numcat" AS "result"
                                                                   WHERE "q12_3" IS DISTINCT
                                                                     FROM TRUE)) AS "ifresult10"

                                                     WHERE NOT "q8_2" IS DISTINCT
                                                       FROM TRUE)
                                                    UNION ALL
                                                     (SELECT "ifresult15".*
                                                      FROM LATERAL
                                                        (SELECT TRUE AS "q12_3") AS "let14"("q12_3"), LATERAL (
                                                                 (SELECT ("numweb_6" + "numstore") + "numcat_4" AS "result"
                                                                  FROM LATERAL
                                                                    (SELECT
                                                                       (SELECT sum("catalog_sales"."cs_net_paid_inc_ship_tax") AS "sum"
                                                                        FROM catalog_sales AS "catalog_sales"
                                                                        WHERE "catalog_sales"."cs_bill_customer_sk" = "c_customer_sk") AS "numcat_4") AS "let16"("numcat_4")
                                                                  WHERE NOT "q12_3" IS DISTINCT
                                                                    FROM TRUE)
                                                                 UNION ALL
                                                                  (SELECT ("numweb_6" + "numstore") + "numcat" AS "result"
                                                                   WHERE "q12_3" IS DISTINCT
                                                                     FROM TRUE)) AS "ifresult15"

                                                      WHERE "q8_2" IS DISTINCT
                                                        FROM TRUE)) AS "ifresult7"

                                      WHERE NOT "q4_1" IS DISTINCT
                                        FROM TRUE)
                                     UNION ALL
                                      (SELECT "ifresult20".*
                                       FROM LATERAL
                                         (SELECT TRUE AS "q8_2") AS "let19"("q8_2"), LATERAL (
                                                    (SELECT "ifresult23".*
                                                     FROM LATERAL
                                                       (SELECT
                                                          (SELECT sum("store_sales"."ss_net_paid_inc_tax") AS "sum"
                                                           FROM store_sales AS "store_sales"
                                                           WHERE "store_sales"."ss_customer_sk" = "c_customer_sk") AS "numstore_5") AS "let21"("numstore_5"), LATERAL
                                                       (SELECT TRUE AS "q12_3") AS "let22"("q12_3"), LATERAL (
                                                                 (SELECT ("numweb" + "numstore_5") + "numcat_4" AS "result"
                                                                  FROM LATERAL
                                                                    (SELECT
                                                                       (SELECT sum("catalog_sales"."cs_net_paid_inc_ship_tax") AS "sum"
                                                                        FROM catalog_sales AS "catalog_sales"
                                                                        WHERE "catalog_sales"."cs_bill_customer_sk" = "c_customer_sk") AS "numcat_4") AS "let24"("numcat_4")
                                                                  WHERE NOT "q12_3" IS DISTINCT
                                                                    FROM TRUE)
                                                                 UNION ALL
                                                                  (SELECT ("numweb" + "numstore_5") + "numcat" AS "result"
                                                                   WHERE "q12_3" IS DISTINCT
                                                                     FROM TRUE)) AS "ifresult23"

                                                     WHERE NOT "q8_2" IS DISTINCT
                                                       FROM TRUE)
                                                    UNION ALL
                                                     (SELECT "ifresult28".*
                                                      FROM LATERAL
                                                        (SELECT TRUE AS "q12_3") AS "let27"("q12_3"), LATERAL (
                                                                 (SELECT ("numweb" + "numstore") + "numcat_4" AS "result"
                                                                  FROM LATERAL
                                                                    (SELECT
                                                                       (SELECT sum("catalog_sales"."cs_net_paid_inc_ship_tax") AS "sum"
                                                                        FROM catalog_sales AS "catalog_sales"
                                                                        WHERE "catalog_sales"."cs_bill_customer_sk" = "c_customer_sk") AS "numcat_4") AS "let29"("numcat_4")
                                                                  WHERE NOT "q12_3" IS DISTINCT
                                                                    FROM TRUE)
                                                                 UNION ALL
                                                                  (SELECT ("numweb" + "numstore") + "numcat" AS "result"
                                                                   WHERE "q12_3" IS DISTINCT
                                                                     FROM TRUE)) AS "ifresult28"

                                                      WHERE "q8_2" IS DISTINCT
                                                        FROM TRUE)) AS "ifresult20"

                                       WHERE "q4_1" IS DISTINCT
                                         FROM TRUE)) AS "ifresult4")
FROM customer;
```

…s most queries.

…es just fine, it unfortunately degenerates …bout this by Sam Arch, who translated …r to our original example in Figure 1, it …nside each other. We show a variation[2]

…nefficient

…ated this work, which is procbench …G22]: The unnesting strategy from …new top-down unnesting Umbra

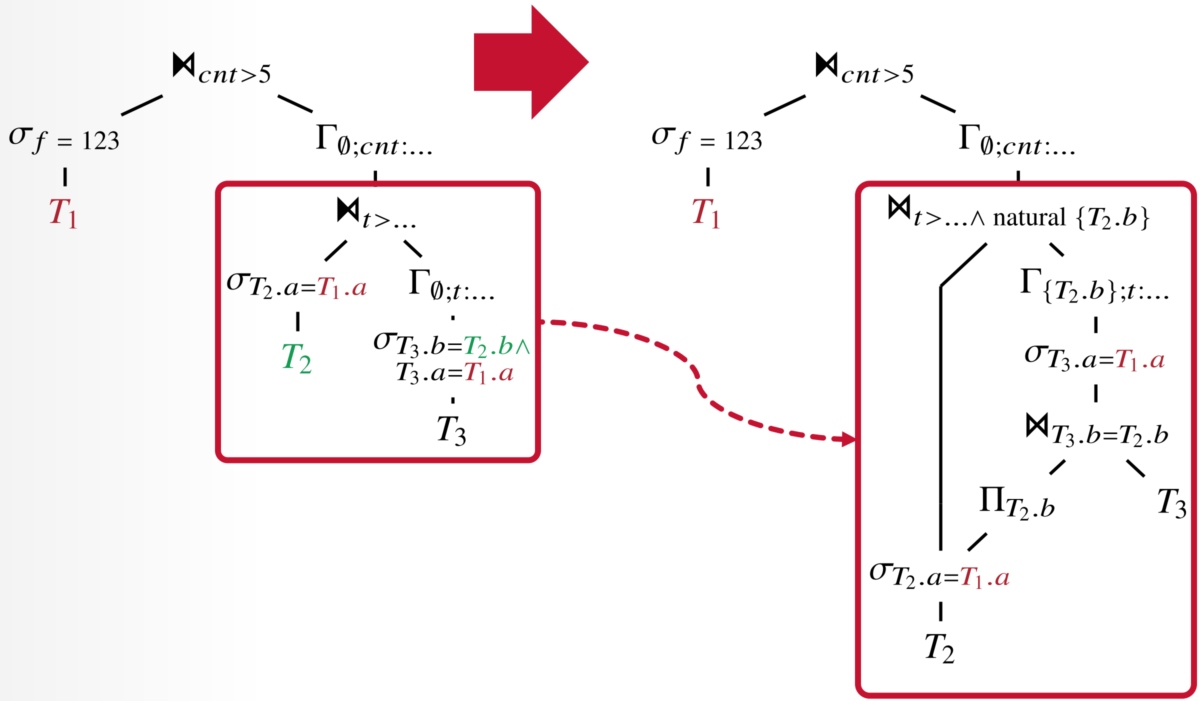
```
SELECT
  (SELECT "ifresult4".*
   FROM LATERAL
     (SELECT NULL :: numeric AS "numweb") AS "let0"("numweb"), LATERAL
     (SELECT NULL :: numeric AS "numstore") AS "let1"("numstore"), LATERAL
     (SELECT NULL :: numeric AS "numcat") AS "let2"("numcat"), LATERAL
     (SELECT TRUE AS "q4_1") AS "let3"("q4_1"), LATERAL (
                    (SELECT "ifresult7".*
                     FROM LATERAL
                       (SELECT
                          (SELECT sum("web_sales"."ws_net_paid_inc_ship_tax") AS "sum"
                           FROM web_sales AS "web_sales"
                           WHERE "web_sales"."ws_bill_customer_sk" = "c_customer_sk") AS "numweb_6") AS "let5"("numweb_6"), LATERAL
                       (SELECT TRUE AS "q8_2") AS "let6"("q8_2"), LATERAL (
                                    (SELECT "ifresult10".*
                                     FROM LATERAL
                                       (SELECT
                                          (SELECT sum("store_sales"."ss_net_paid_inc_tax") AS "sum"
                                           FROM store_sales AS "store_sales"
                                           WHERE "store_sales"."ss_customer_sk" = "c_customer_sk") AS "numstore_5") AS "let8"("numstore_5"), LATERAL
                                       (SELECT TRUE AS "q12_3") AS "let9"("q12_3"), LATERAL (
                                                    (SELECT ("numweb_6" + "numstore_5") + "numcat_4" AS "result"
                                                     FROM LATERAL
                                                       (SELECT
                                                          (SELECT sum("catalog_sales"."cs_net_paid_inc_ship_tax") AS "sum"
                                                           FROM catalog_sales AS "catalog_sales"
                                                           WHERE "catalog_sales"."cs_bill_customer_sk" = "c_customer_sk") AS "numcat_4") AS "let11"("numcat_4")
                                                     WHERE NOT "q12_3" IS DISTINCT
                                                       FROM TRUE)
                                                   UNION ALL
                                                    (SELECT ("numweb_6" + "numstore_5") + "numcat" AS "result"
                                                     WHERE "q12_3" IS DISTINCT
                                                       FROM TRUE)) AS "ifresult10"
                                     WHERE NOT "q8_2" IS DISTINCT
                                       FROM TRUE)
                                   UNION ALL
                                    (SELECT "ifresult15".*
                                     FROM LATERAL
                                       (SELECT TRUE AS "q12_3") AS "let14"("q12_3"), LATERAL (
                                                    (SELECT ("numweb_6" + "numstore") + "numcat_4" AS "result"
                                                     FROM LATERAL
                                                       (SELECT
                                                          (SELECT sum("catalog_sales"."cs_net_paid_inc_ship_tax") AS "sum"
                                                           FROM catalog_sales AS "catalog_sales"
                                                           WHERE "catalog_sales"."cs_bill_customer_sk" = "c_customer_sk") AS "numcat_4") AS "let16"("numcat_4")
                                                     WHERE NOT "q12_3" IS DISTINCT
                                                       FROM TRUE)
                                                   UNION ALL
                                                    (SELECT ("numweb_6" + "numstore") + "numcat" AS "result"
                                                     WHERE "q12_3" IS DISTINCT
                                                       FROM TRUE)) AS "ifresult15"
                                     WHERE "q8_2" IS DISTINCT
                                       FROM TRUE)) AS "ifresult7"
...
                                                       FROM TRUE)
                                                   UNION ALL
                                                    (SELECT ("numweb" + "numstore") + "numcat" AS "result"
                                                     WHERE "q12_3" IS DISTINCT
                                                       FROM TRUE)) AS "ifresult28"
                                     WHERE "q8_2" IS DISTINCT
                                       FROM TRUE)) AS "ifresult20"
                WHERE "q4_1" IS DISTINCT
                  FROM TRUE)) AS "ifresult4")
FROM customer;
```

```
(SELECT "ifresult7".*
 FROM LATERAL
   (SELECT
      (SELECT sum("web_sales"."ws_net_paid_inc_ship_tax") AS "sum"
       FROM web_sales AS "web_sales"
       WHERE "web_sales"."ws_bill_customer_sk" = "c_customer_sk") AS "numweb_6") AS "let5"("numweb_6"), LATERAL
   (SELECT TRUE AS "q8_2") AS "let6"("q8_2"), LATERAL (
                                        (SELECT "ifresult10".*
                                         FROM LATERAL
                                           (SELECT
                                              (SELECT sum("store_sales"."ss_net_paid_inc_tax") AS "sum"
                                               FROM store_sales AS "store_sales"
                                               WHERE "store_sales"."ss_customer_sk" = "c_customer_sk") AS "numstore_5") AS "let8"("numstore_5"), LATERAL
                                           (SELECT TRUE AS "q12_3") AS "let9"("q12_3"), LATERAL (
                                                                                  (SELECT ("numweb_6" + "numstore_5") + "numcat_4" AS "result"
```

s most queries.

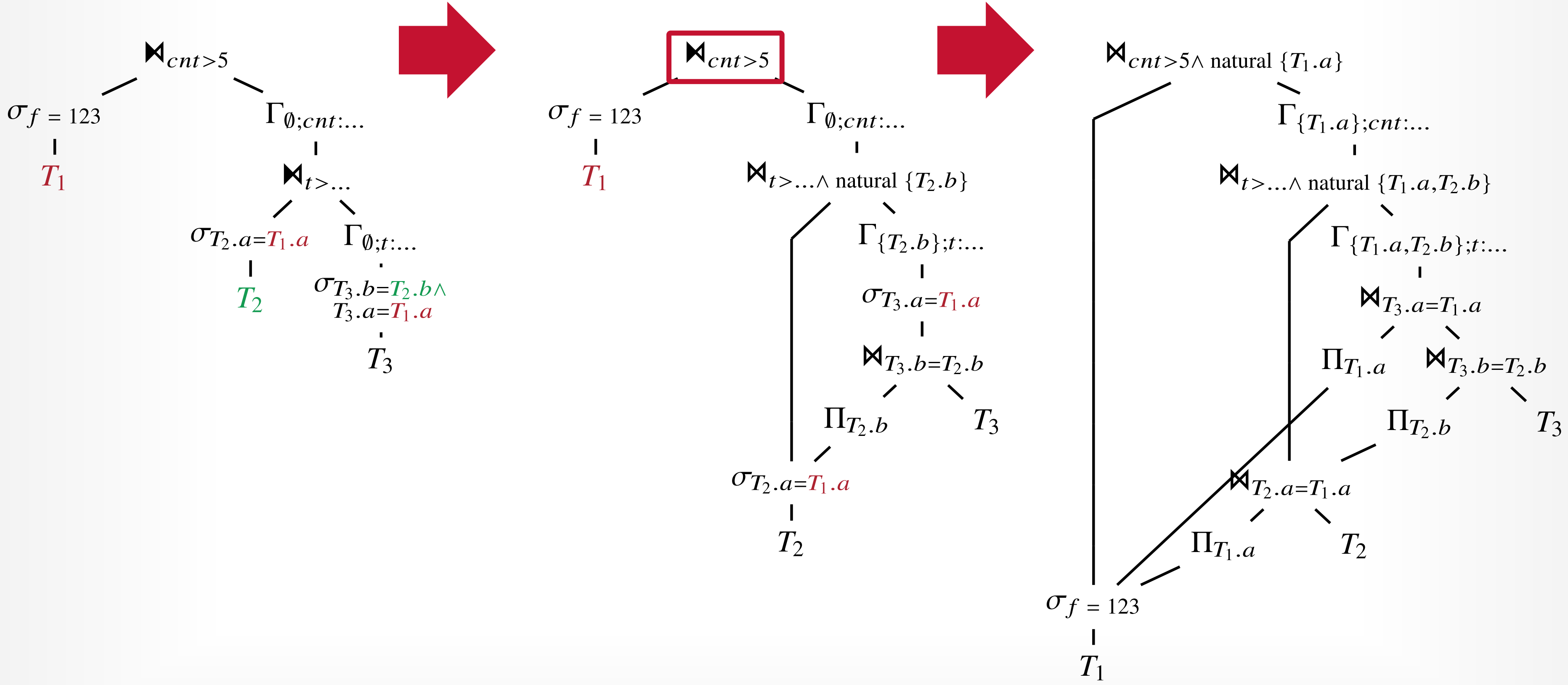
s just fine, it unfortunately degenerates
bout this by Sam Arch, who translated
r to our original example in Figure 1, it
nside each other. We show a variation[2]

CRASH.SQL



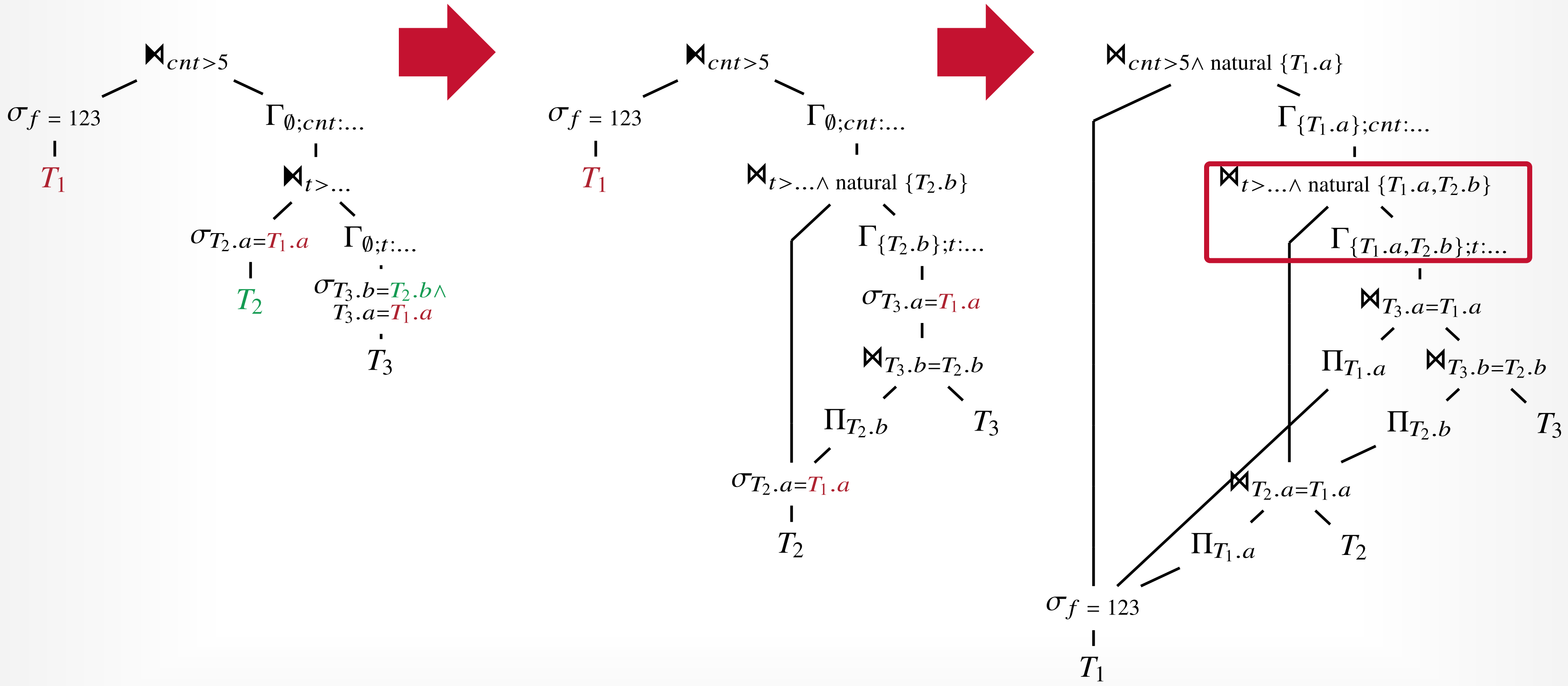
Source: Thomas Neumann

CRASH.SQL



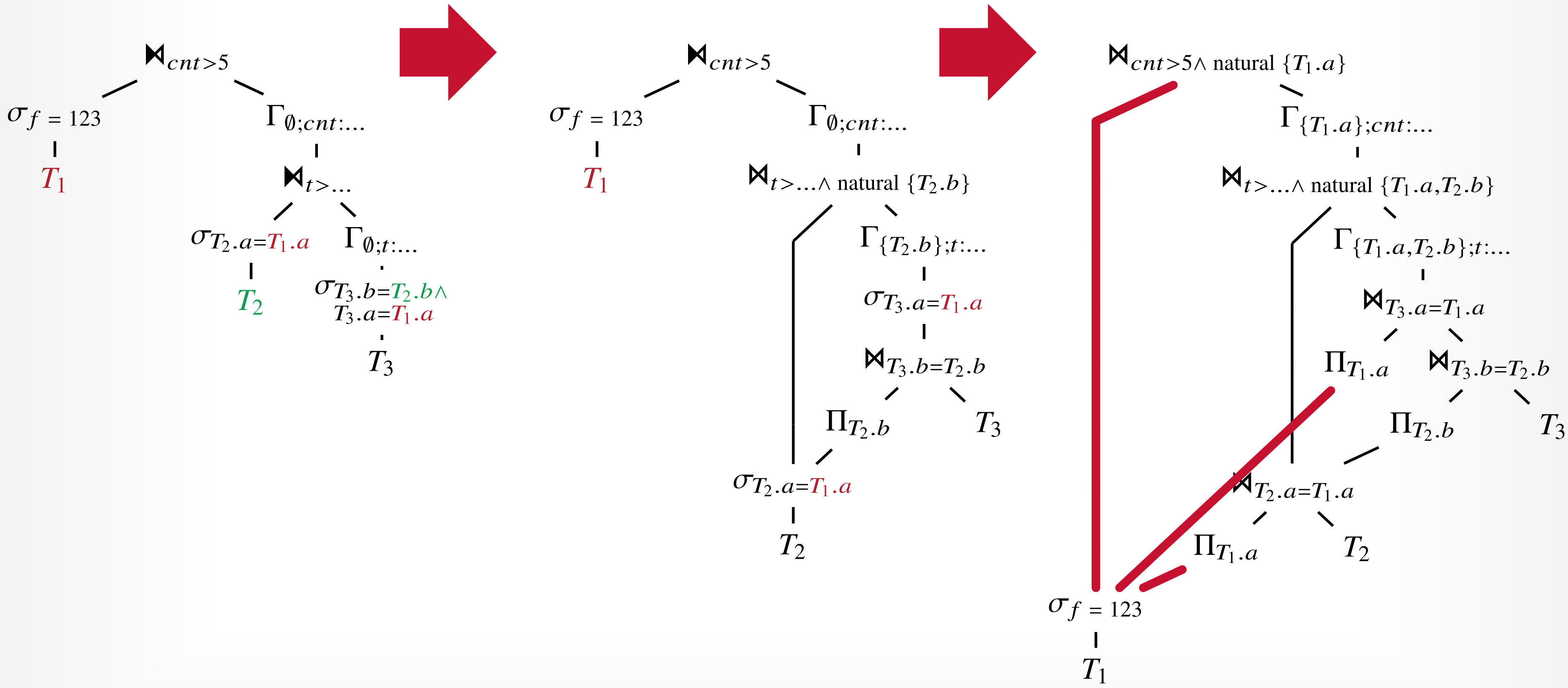
Source: Thomas Neumann

CRASH.SQL



Source: Thomas Neumann

CRASH.SQL



Source: Thomas Neumann

HOLISTIC UNNESTING (2025)

Remove all dependent joins at the same time starting at the top of the query plan.

→ Keep track of where they are in the plan and then rewrite all operators in a top-down pass until each join is unnecessary or it can be safely added.
→ Avoids pushing dependency sets across joins.

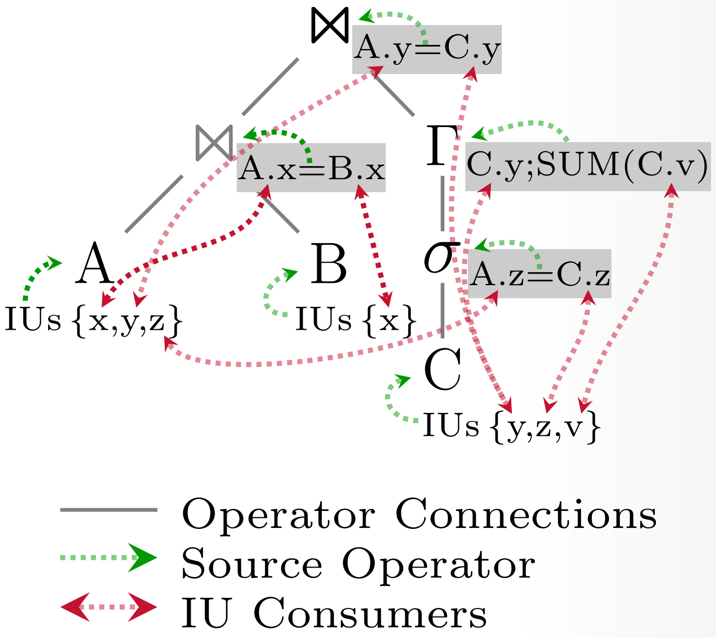
The optimizer needs an efficient way to identify the flow of attributes through the plan…

IMPROVING UNNESTING OF COMPLEX QUERIES
BTW 2025

INDEXED ALGEBRA

Unnesting subqueries requires the optimizer to reason about the dependencies and flow of attributes in a query plan's operators.

Maintain an auxiliary index of operator meta-data to facilitate faster examination of plans and to identify rewrite opportunities.

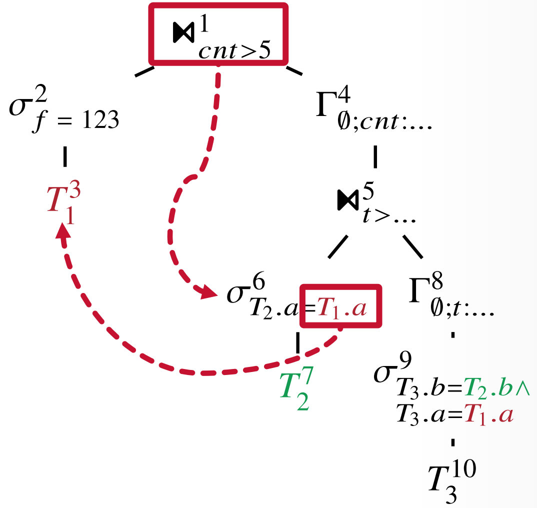


ASYMPTOTICALLY BETTER QUERY OPTIMIZATION USING INDEXED ALGEBRA
VLDB 2023

HOLISTIC UNNESTING: IDENTIFICATION

Identify dependent joins where the RHS accesses attributes provided by the LHS.

For each column accessed, compute the <u>lowest common ancestor</u> of operator o_1 that accesses a column and operator o_2 that provides the column.
→ If $o_1 \neq o_2$, then it is a dependent join.

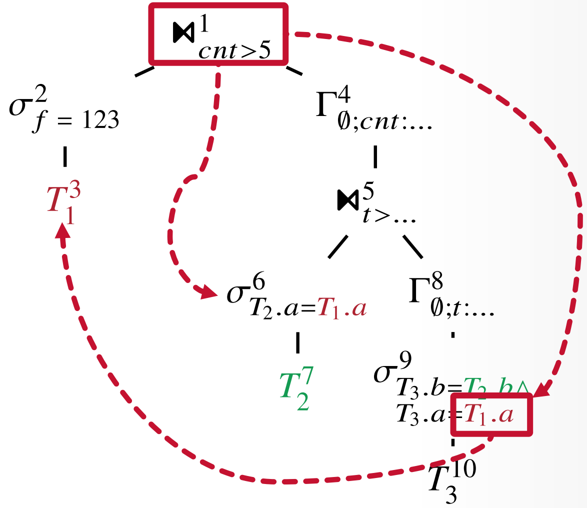


Source: Thomas Neumann

HOLISTIC UNNESTING: IDENTIFICATION

Identify dependent joins where the RHS accesses attributes provided by the LHS.

For each column accessed, compute the lowest common ancestor of operator o_1 that accesses a column and operator o_2 that provides the column.
→ If $o_1 \neq o_2$, then it is a dependent join.



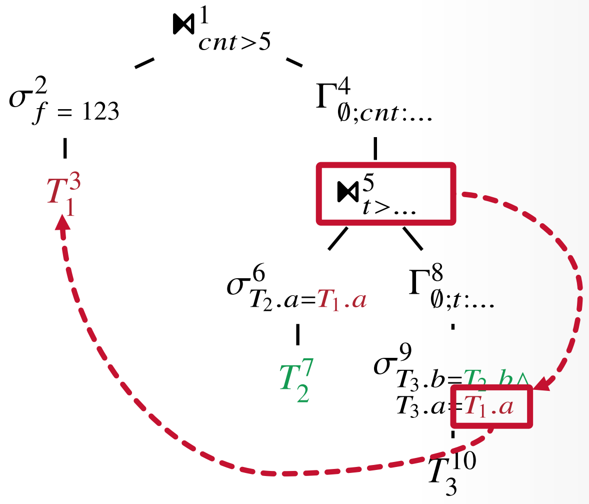
$$\text{accessing}(\bowtie^1) := \{\sigma^6, \sigma^9\}$$

Source: Thomas Neumann

HOLISTIC UNNESTING: IDENTIFICATION

Identify dependent joins where the RHS accesses attributes provided by the LHS.

For each column accessed, compute the lowest common ancestor of operator o_1 that accesses a column and operator o_2 that provides the column.
→ If $o_1 \neq o_2$, then it is a dependent join.



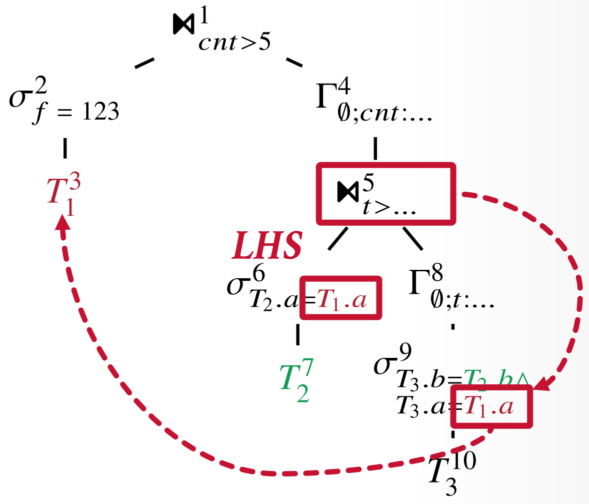
accessing($\bowtie^1$) := $\{\sigma^6, \sigma^9\}$
accessing($\bowtie^5$) := $\{\sigma^9\}$

Source: Thomas Neumann

HOLISTIC UNNESTING: IDENTIFICATION

Identify dependent joins where the RHS accesses attributes provided by the LHS.

For each column accessed, compute the lowest common ancestor of operator ***o₁*** that accesses a column and operator ***o₂*** that provides the column.
→ If $o_1 \neq o_2$, then it is a dependent join.



$$\text{accessing}(⋈^1) := \{\sigma^6, \sigma^9\}$$
$$\text{accessing}(⋈^5) := \{\sigma^9\}$$

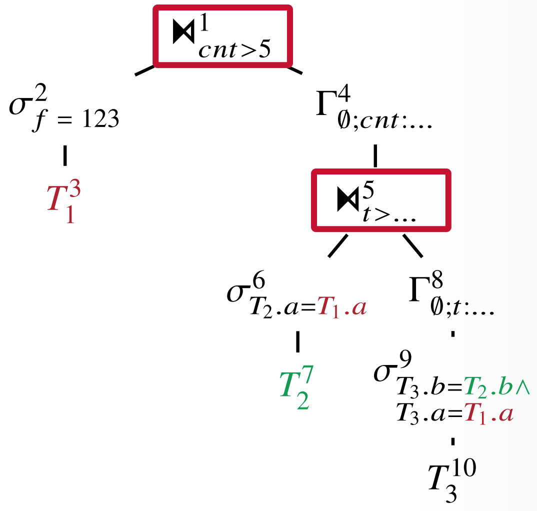
Source: Thomas Neumann

SIMPLE ELIMINATION

Inspect all operators that access the LHS of a dependent join.

Then use the "simple" dependent join elimination discussed earlier.
→ Move operators up towards the join.

Otherwise, use the full unnesting algorithm…

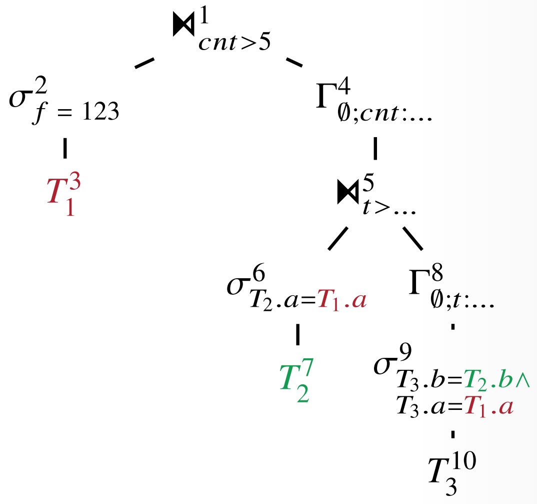


HOLISTIC ELIMINATION

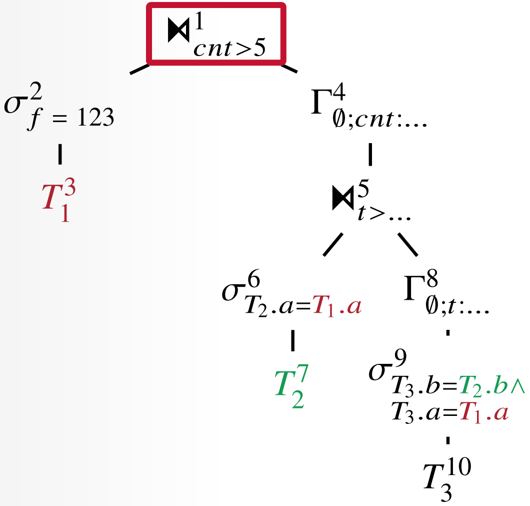
Rewrite RHS of dependent join such that no references from the "outer" side occur anymore.

→ Columns from the LHS that are accessed from the RHS.

Maintain state about the algorithm's progress to keep track of where columns are coming from in plan.



HOLISTIC ELIMINATION

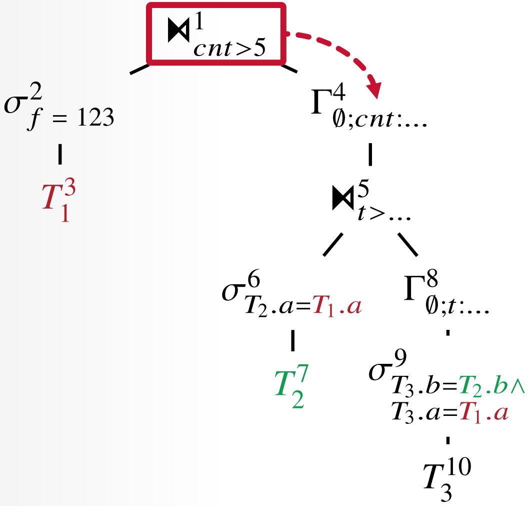


outerRef:=$\{T_1.a\}$
cclasses:=$\emptyset$
repr:=$\emptyset$

next: unnest(Γ^4, $\{\sigma^6, \sigma^9\}$)

Source: Thomas Neumann

HOLISTIC ELIMINATION

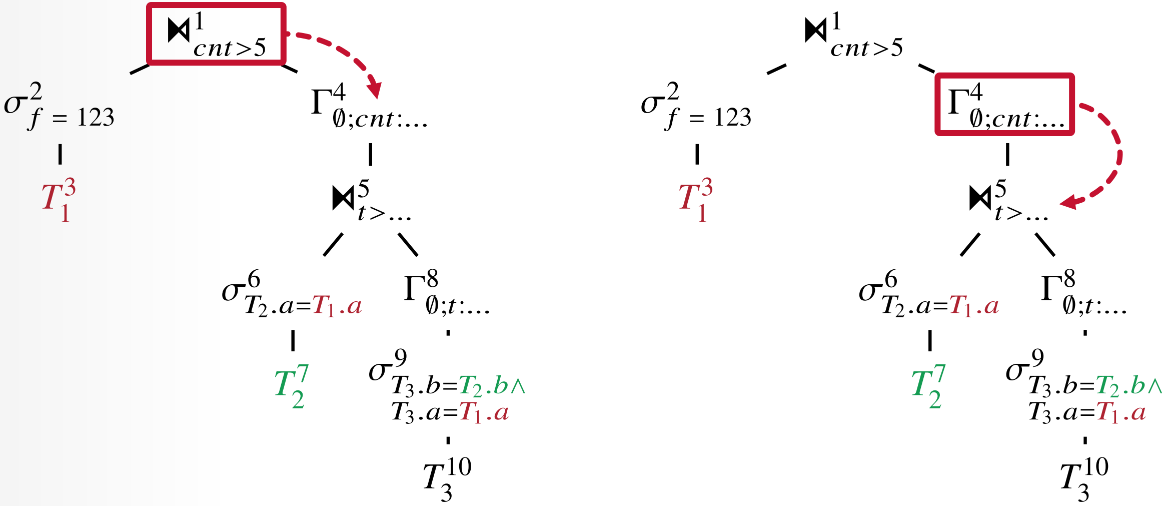


outerRef:=$\{T_1.a\}$
cclasses:=$\emptyset$
repr:=$\emptyset$

next: unnest(Γ^4, $\{\sigma^6, \sigma^9\}$)

Source: Thomas Neumann

HOLISTIC ELIMINATION



outerRef:={$T_1.a$}
cclasses:=∅
repr:=∅

next: unnest(Γ^4, {σ^6, σ^9})

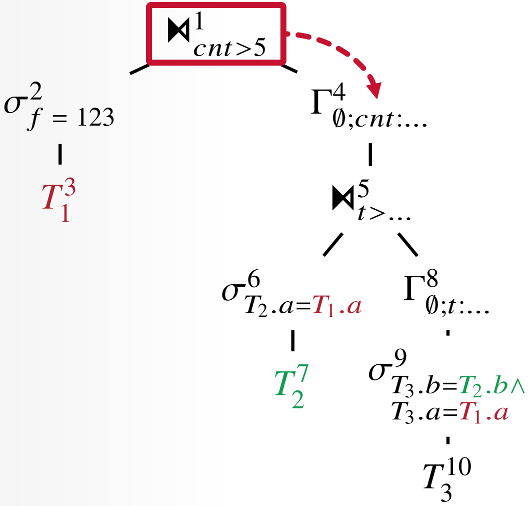
⇒

outerRef:={$T_1.a$}
cclasses:=∅
repr:=∅

stack: [Γ^4]
next: unnest($\bowtie^5$, {σ^6, σ^9})

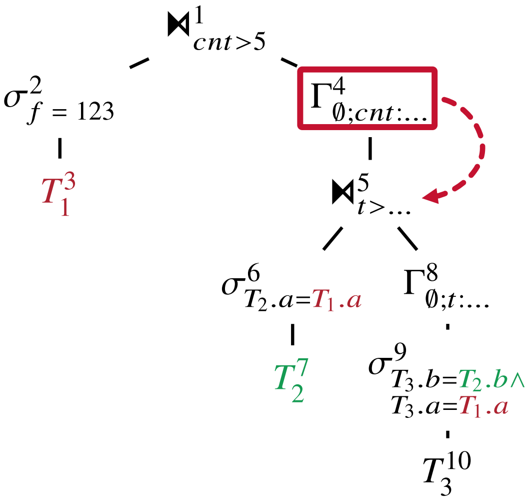
Source: Thomas Neumann

HOLISTIC ELIMINATION



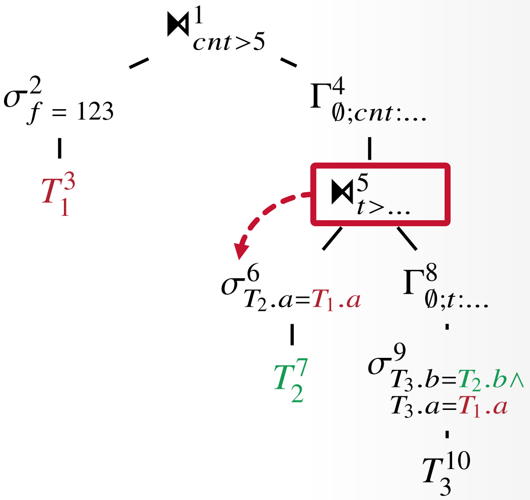
outerRef:=$\{T_1.a\}$
cclasses:=$\emptyset$
repr:=$\emptyset$

next: unnest(Γ^4, $\{\sigma^6, \sigma^9\}$)



outerRef:=$\{T_1.a\}$
cclasses:=$\emptyset$
repr:=$\emptyset$

stack: [Γ^4]
next: unnest($\bowtie^5$, $\{\sigma^6, \sigma^9\}$)

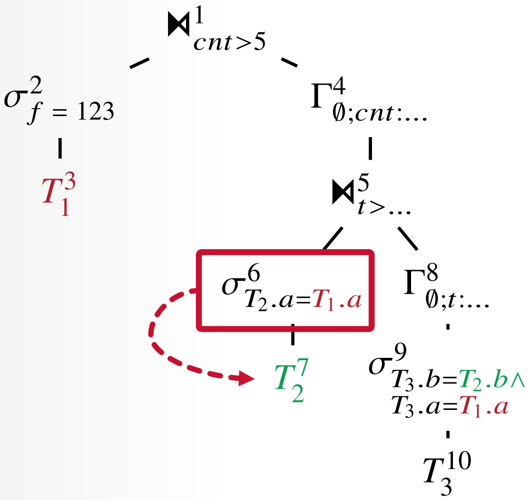


outerRef:=$\{T_1.a\}$
cclasses:=$\{\{T_1.a, T_2.a\}\}$
repr:=$\emptyset$

stack: [Γ^4, $\bowtie^5$]
next: unnest(σ^6, $\{\sigma^6\}$)

Source: Thomas Neumann

HOLISTIC ELIMINATION

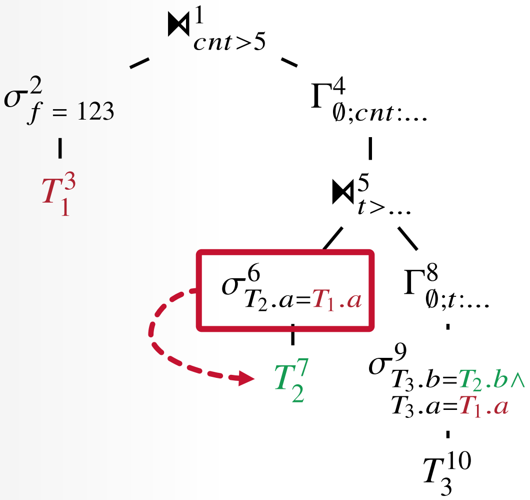


outerRef:=$\{T_1.a\}$
cclasses:=$\{\{T_1.a, T_2.a\}\}$
repr:=$\emptyset$

stack: $[\Gamma^4, \bowtie^5, \sigma^6]$
next: unnest(T^7_2, {})

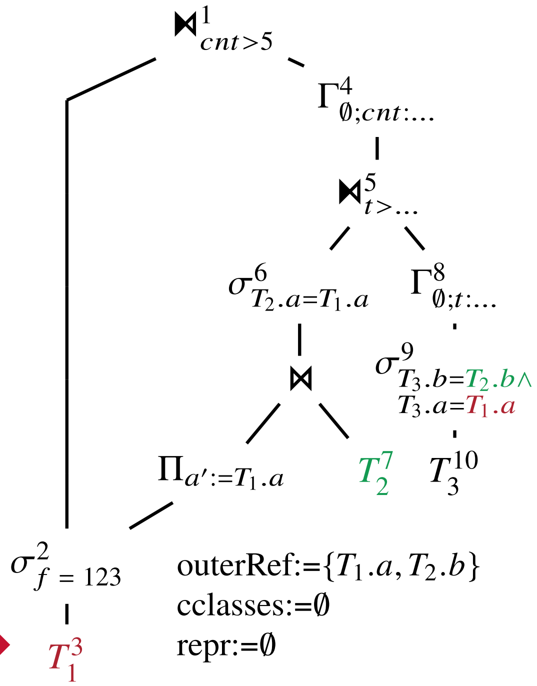
Source: Thomas Neumann

HOLISTIC ELIMINATION



outerRef:=$\{T_1.a\}$
cclasses:=$\{\{T_1.a, T_2.a\}\}$
repr:=$\emptyset$

stack: $[\Gamma^4, \bowtie^5, \sigma^6]$
next: unnest(T^7_2, $\{\}$)

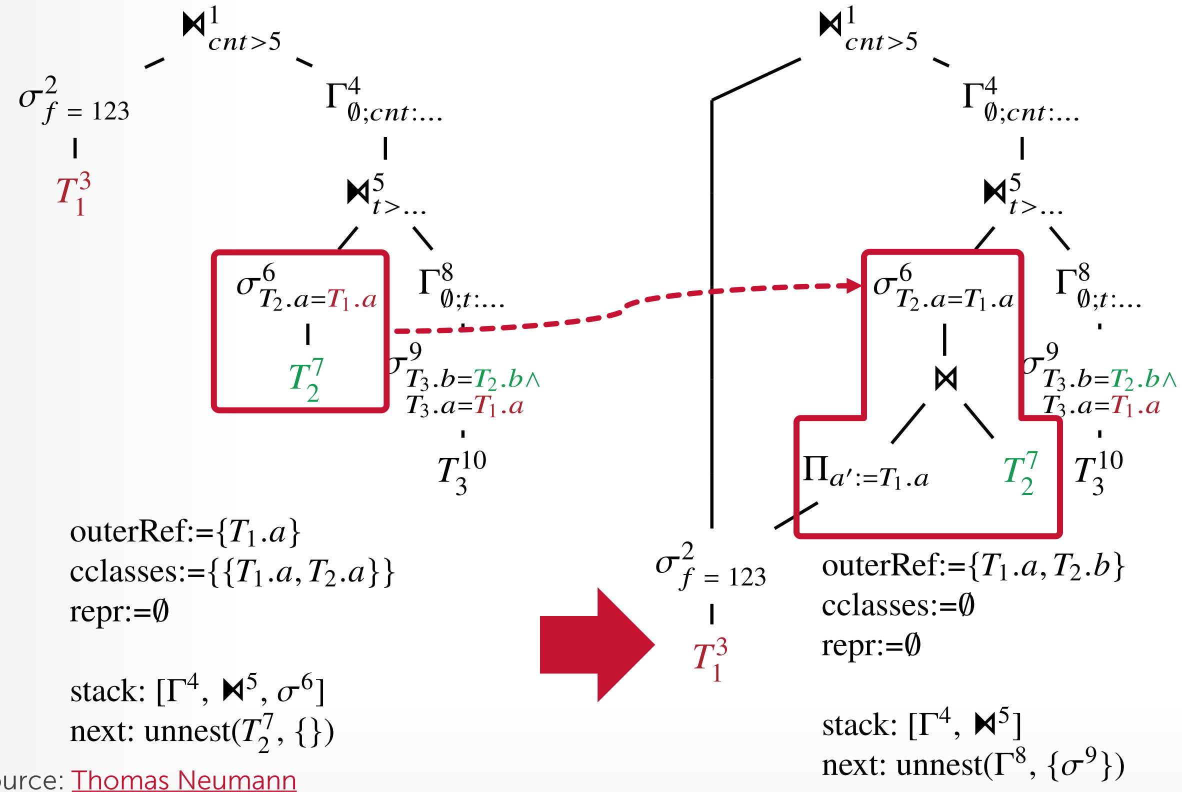


outerRef:=$\{T_1.a, T_2.b\}$
cclasses:=$\emptyset$
repr:=$\emptyset$

stack: $[\Gamma^4, \bowtie^5]$
next: unnest(Γ^8, $\{\sigma^9\}$)

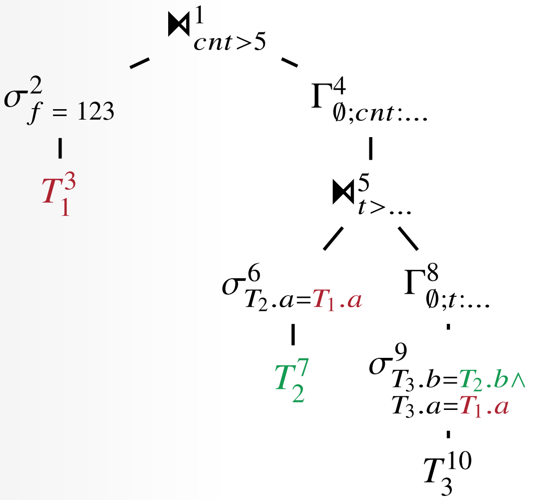
Source: Thomas Neumann

HOLISTIC ELIMINATION



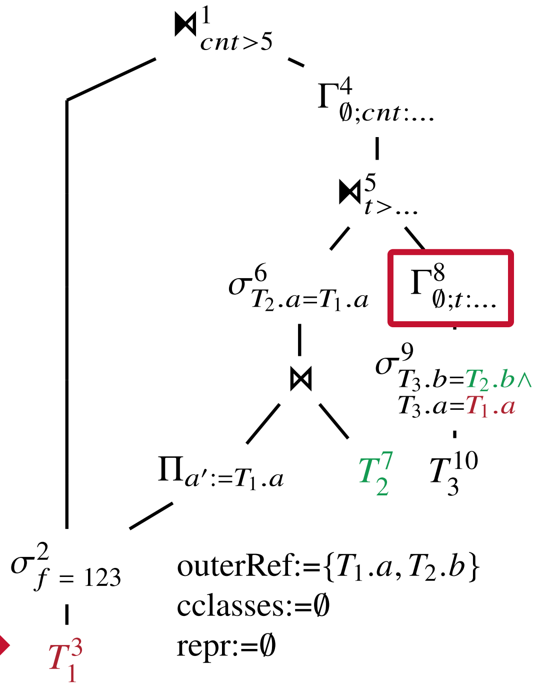
Source: Thomas Neumann

HOLISTIC ELIMINATION



outerRef:=$\{T_1.a\}$
cclasses:=$\{\{T_1.a, T_2.a\}\}$
repr:=$\emptyset$

stack: $[\Gamma^4, \bowtie^5, \sigma^6]$
next: unnest(T^7_2, {})

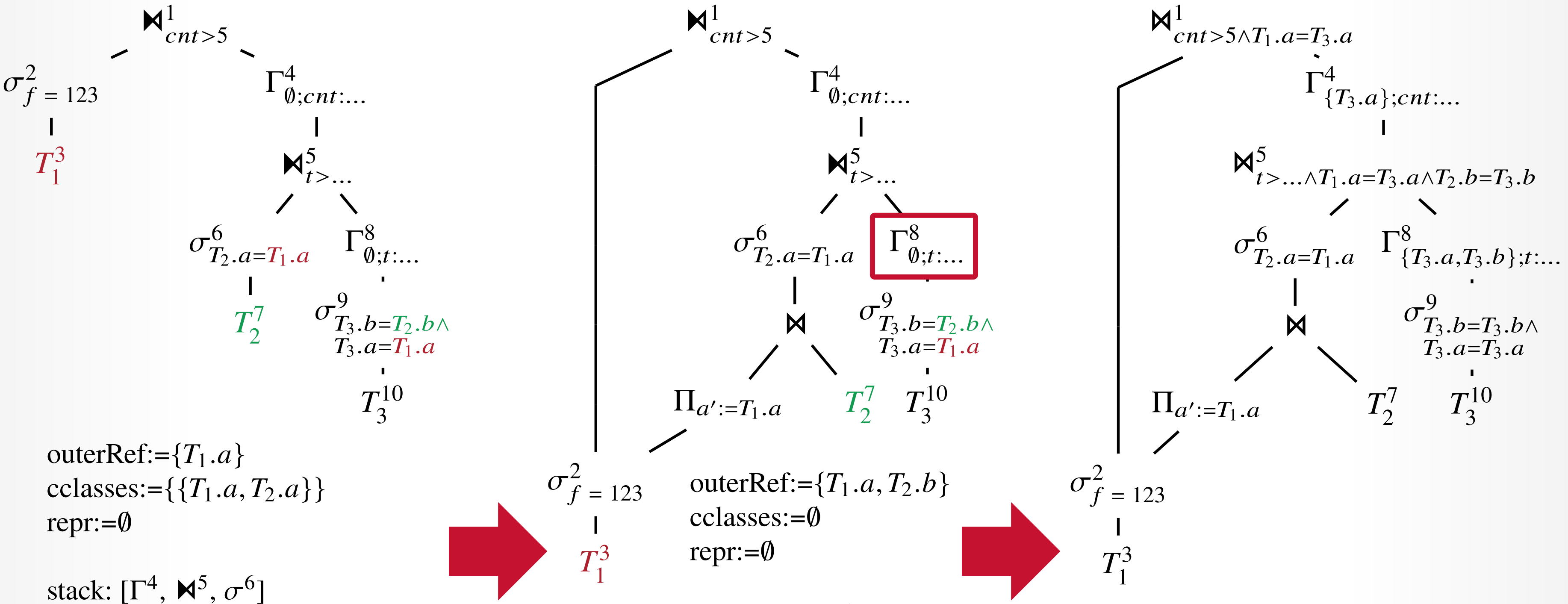


outerRef:=$\{T_1.a, T_2.b\}$
cclasses:=$\emptyset$
repr:=$\emptyset$

stack: $[\Gamma^4, \bowtie^5]$
next: unnest(Γ^8, $\{\sigma^9\}$)

Source: Thomas Neumann

HOLISTIC ELIMINATION



outerRef:=$\{T_1.a\}$
cclasses:=$\{\{T_1.a, T_2.a\}\}$
repr:=$\emptyset$

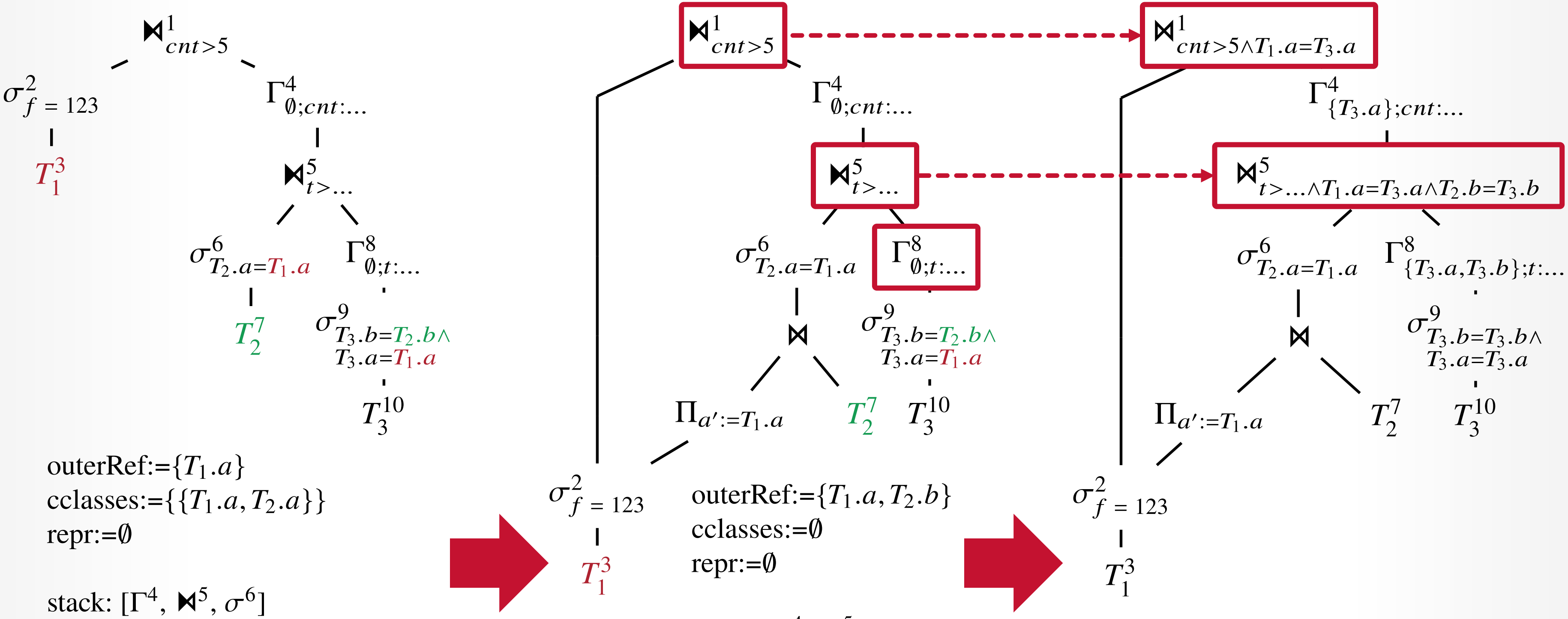
stack: $[\Gamma^4, \bowtie^5, \sigma^6]$
next: unnest(T^7_2, {})

outerRef:=$\{T_1.a, T_2.b\}$
cclasses:=$\emptyset$
repr:=$\emptyset$

stack: $[\Gamma^4, \bowtie^5]$
next: unnest(Γ^8, $\{\sigma^9\}$)

Source: Thomas Neumann

HOLISTIC ELIMINATION



outerRef:= $\{T_1.a\}$
cclasses:= $\{\{T_1.a, T_2.a\}\}$
repr:= $\emptyset$

stack: $[\Gamma^4, ⋈^5, \sigma^6]$
next: unnest(T_2^7, {})

outerRef:= $\{T_1.a, T_2.b\}$
cclasses:= $\emptyset$
repr:= $\emptyset$

stack: $[\Gamma^4, ⋈^5]$
next: unnest(Γ^8, $\{\sigma^9\}$)

Source: Thomas Neumann

PARTING THOUGHTS

Holistic unnesting is the definitive way to decorrelate subqueries.

→ Relies on DBMS supporting DAG query plans.

→ Build indexes to speed up query plan analysis during optimization phases.

We will see correlated subqueries again when discussing UDF inlining.

NEXT CLASS

Cost Models! Statistics!

→ aka when everything falls apart…