Universität Konstanz

Optimization of Nested Queries using the NF² Algebra

Jürgen Hölsch, Michael Grossniklaus, and Marc H. Scholl

ACM SIGMOD Conference, June 30, 2016

Motivation

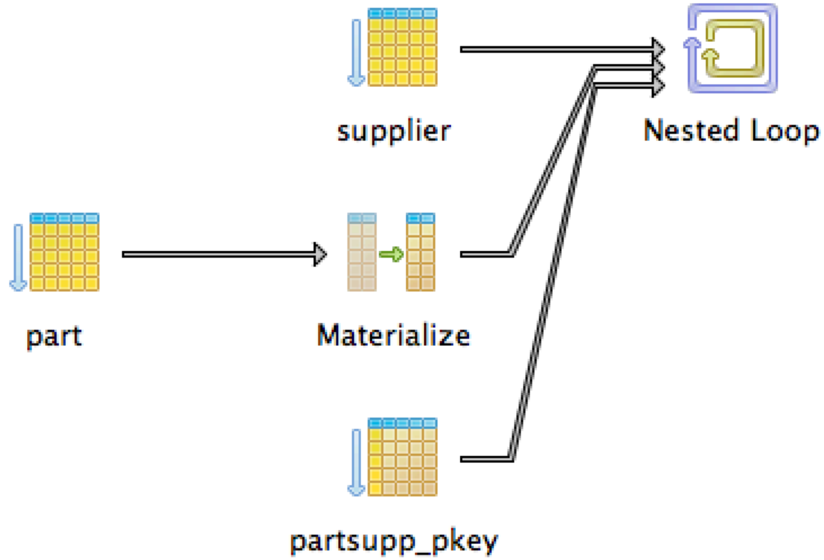
Example

"*All Parts offered by a supplier*"

```
SELECT p_name, s_name
FROM Part, Supplier
WHERE p_partkey IN (SELECT ps_partkey
                    FROM PartSupp
                    WHERE ps_suppkey = s_suppkey)
```

Motivation

- Execution plan in PostgreSQL 9.4.2:



- Execution time on a 10 GB TPC-H DB: $>$ 24 h

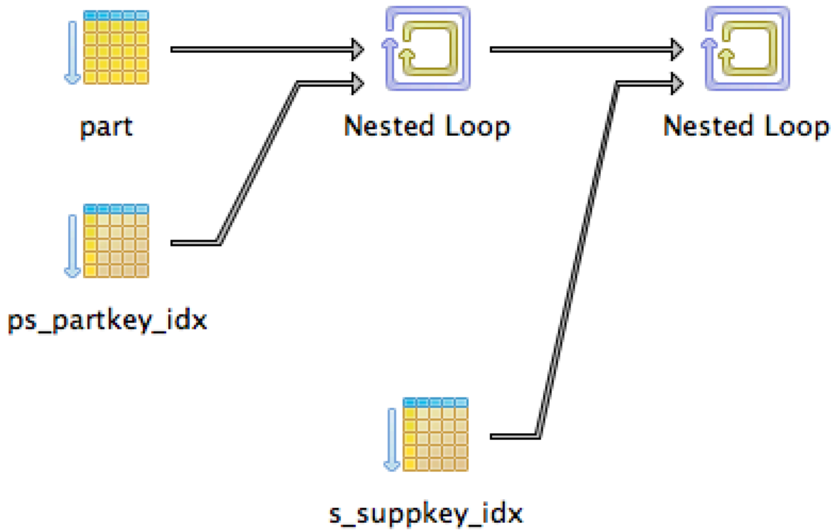
Motivation

Alternative formulation

```
SELECT p_name, s_name
FROM Part, Supplier, PartSupp
WHERE p_partkey = ps_partkey AND ps_suppkey = s_suppkey
```

Motivation

- Execution plan in PostgreSQL 9.4.2:



- Execution time on a 10 GB TPC-H DB: $\approx$ 32 s

Techniques for optimizing nested queries

- Transformations at the level of SQL
- Different formalisms (*e.g.* comprehension calculus)

Techniques for optimizing nested queries

- Transformations at the level of SQL
  $\Rightarrow$ Separates nested query optimization from other optimization steps

- Different formalisms (*e.g.* comprehension calculus)

Techniques for optimizing nested queries

- Transformations at the level of SQL
  $\Rightarrow$ Separates nested query optimization from other optimization steps
- Different formalisms (*e.g.* comprehension calculus)
  $\Rightarrow$ Not used in real world optimizers

Techniques for optimizing nested queries

- Transformations at the level of SQL
  $\Rightarrow$ Separates nested query optimization from other optimization steps

- Different formalisms (*e.g.* comprehension calculus)
  $\Rightarrow$ Not used in real world optimizers

$\rightarrow$ Ideally, handle nested query optimization **algebraically**

How can we optimize this query algebraically?

Example

```
SELECT p_name, s_name
FROM Part, Supplier
WHERE p_partkey IN (SELECT ps_partkey
                    FROM PartSupp
                    WHERE ps_suppkey = s_suppkey)
```

How can we optimize this query algebraically?

Example

```
SELECT p_name, s_name
FROM Part, Supplier
WHERE p_partkey IN (SELECT ps_partkey
                    FROM PartSupp
                    WHERE ps_suppkey = s_suppkey)
```

Let's try to use the relational algebra:

How can we optimize this query algebraically?

Example

```
SELECT p_name, s_name
FROM Part, Supplier
WHERE p_partkey IN (SELECT ps_partkey
                    FROM PartSupp
                    WHERE ps_suppkey = s_suppkey)
```

Let's try to use the relational algebra:

$$\pi[\text{p\_name}, \text{s\_name}](\qquad\qquad\qquad\qquad)$$

How can we optimize this query algebraically?

Example

$\texttt{SELECT}$ p_name, s_name
$\texttt{FROM}$ Part, Supplier
$\texttt{WHERE}$ p_partkey $\texttt{IN}$ ($\texttt{SELECT}$ ps_partkey
$\texttt{FROM}$ PartSupp
$\texttt{WHERE}$ ps_suppkey $=$ s_suppkey)

Let's try to use the relational algebra:

$$\pi[\text{p\_name, s\_name}](\quad \text{Part} \times \text{Supplier}\ )$$

How can we optimize this query algebraically?

Example

SELECT p_name, s_name
FROM Part, Supplier
WHERE p_partkey IN (SELECT ps_partkey
                    FROM PartSupp
                    WHERE ps_suppkey = s_suppkey)

Let's try to use the relational algebra:

$$\pi[\text{p\_name, s\_name}](\sigma[?](\text{Part} \times \text{Supplier}))$$

How can we optimize this query algebraically?

Example

```
SELECT p_name, s_name
FROM Part, Supplier
WHERE p_partkey IN (SELECT ps_partkey
                    FROM PartSupp
                    WHERE ps_suppkey = s_suppkey)
```

~~Let's try to use the relational algebra:~~

$$\cancel{\pi[\text{p\_name, s\_name}](\sigma[?](\text{Part} \times \text{Supplier}))}$$

Nestings cannot be represented by the relational algebra

How can we optimize this query algebraically?

$\Rightarrow$ Representation with NF^2 algebra

Example

$\texttt{SELECT}$ p_name, s_name
$\texttt{FROM}$ Part, Supplier
$\texttt{WHERE}$ p_partkey $\texttt{IN}$ ($\texttt{SELECT}$ ps_partkey
$\texttt{FROM}$ PartSupp
$\texttt{WHERE}$ ps_suppkey $=$ s_suppkey)

NF^2 representation:

$$\pi[\text{p\_name, s\_name}](\sigma[\text{p\_partkey} \in \pi[\text{ps\_partkey}](\sigma[\text{ps\_suppkey} = \text{s\_suppkey}](\text{PartSupp}))](\text{Part} \times \text{Supplier}))$$

How can we optimize this query algebraically?

$\Rightarrow$ Representation with NF^2 algebra

Example

```
SELECT p_name, s_name
FROM Part, Supplier
WHERE p_partkey IN (SELECT ps_partkey
                    FROM PartSupp
                    WHERE ps_suppkey = s_suppkey)
```

NF^2 representation:

$$\pi[\text{p\_name, s\_name}](\sigma[\text{p\_partkey} \in \pi[\text{ps\_partkey}](\sigma[\text{ps\_suppkey} = \text{s\_suppkey}](\text{PartSupp}))](\text{Part} \times \text{Supplier}))$$

How can we optimize this query algebraically?

$\Rightarrow$ Representation with NF^2 algebra

No NF^2 backend is needed

Example

```
SELECT p_name, s_name
FROM Part, Supplier
WHERE p_partkey IN (SELECT ps_partkey
                    FROM PartSupp
                    WHERE ps_suppkey = s_suppkey)
```

NF^2 representation:

$$\pi[\text{p\_name}, \text{s\_name}](\sigma[\text{p\_partkey} \in \pi[\text{ps\_partkey}](\sigma[\text{ps\_suppkey} = \text{s\_suppkey}](\text{PartSupp}))](\text{Part} \times \text{Supplier}))$$

Our Contributions

- We show how all types of nested queries are represented by NF^2 expressions

Our Contributions

- We show how all types of nested queries are represented by NF^2 expressions
- We define NF^2 equivalences that formalize existing optimization techniques

Our Contributions

- We show how all types of nested queries are represented by NF^2 expressions
- We define NF^2 equivalences that formalize existing optimization techniques
- We introduce new optimization techniques, which are made possible by the NF^2 approach

Our Contributions

- We show how all types of nested queries are represented by NF^2 expressions
- We define NF^2 equivalences that formalize existing optimization techniques
- We introduce new optimization techniques, which are made possible by the NF^2 approach
- We discuss the neccessary changes to an optimizer based on Cascades framework

Our Contributions

- We show how all types of nested queries are represented by NF^2 expressions
- We define NF^2 equivalences that formalize existing optimization techniques
- We introduce new optimization techniques, which are made possible by the NF^2 approach
- We discuss the neccessary changes to an optimizer based on Cascades framework
- We quantify the performance benefits of our approach

Equivalences for existing optimization techniques

Won Kim, *On Optimizing an SQL-like Nested Query*. ACM Transactions on Database Systems (TODS), 1982.

Unnesting of Type J and N queries

$$\sigma[A \in \pi[B](\sigma[F](Inner))](Outer) \equiv \pi[attr(Outer)](Inner \bowtie_{A=B \wedge F} Outer)$$

Unnesting of Type A queries

$$\sigma[A\ \theta\ f(\pi[B](Inner))](Outer)$$
$$\equiv \pi[attr(Outer)](\sigma[A\ \theta\ agg](Outer \times (agg := f(\pi[B](Inner))))$$

Unnesting of Type JA queries

$$\sigma[A\ \theta\ f(\pi[B](\sigma[F](Inner)))](Outer)$$
$$\equiv \pi[attr(Outer)](\sigma[A\ \theta\ agg](Outer \bowtie_F \gamma[G; agg := f(B)](Inner)))$$

If F has inequality predicates:

$$\sigma[A\ \theta\ f(\pi[B](\sigma[F](Inner)))](Outer)$$
$$\equiv \pi[attr(Outer)](\sigma[A\ \theta\ agg](Outer \bowtie \gamma[G; agg := f(B)](Outer \bowtie_F Inner)))$$

If f is COUNT:

$$\sigma[A\ \theta\ COUNT(\pi[B](\sigma[F](Inner)))](Outer)$$
$$\equiv \pi[attr(Outer)](\sigma[A\ \theta\ agg](Outer \bowtie \gamma[G; agg := COUNT(B)](Outer ⟕ Inner)))$$

Example: Type J and N unnesting rule

Type J and N unnesting rule

$$\sigma[A \in \pi[B](\sigma[F](Inner))](Outer)$$
$$\equiv \pi[attr(Outer)](Inner \bowtie_{A=B \wedge F} Outer)$$

Query from the introduction

$$\pi[\text{p\_name, s\_name}](\sigma[\text{p\_partkey} \in \pi[\text{ps\_partkey}]($$
$$\sigma[\text{ps\_suppkey} = \text{s\_suppkey}](\text{PartSupp}))](\text{Part} \times \text{Supplier}))$$

$$\equiv \pi[\text{p\_name, s\_name}]($$
$$\text{PartSupp} \ltimes_{\text{p\_partkey}=\text{ps\_partkey} \wedge \text{ps\_suppkey}=\text{s\_suppkey}} (\text{Part} \times \text{Supplier}))$$

Example: Type J and N unnesting rule

Existing relational algebra equivalences remain valid

Type J and N unnesting rule

$$\sigma[A \in \pi[B](\sigma[F](Inner))](Outer)$$
$$\equiv \pi[attr(Outer)](Inner \bowtie_{A=B \wedge F} Outer)$$

Query from the introduction

$$\pi[\text{p\_name, s\_name}](\sigma[\text{p\_partkey} \in \pi[\text{ps\_partkey}]($$
$$\sigma[\text{ps\_suppkey} = \text{s\_suppkey}](\text{PartSupp}))](\text{Part} \times \text{Supplier}))$$

$$\equiv \pi[\text{p\_name, s\_name}]($$
$$\text{PartSupp} \bowtie_{\text{p\_partkey}=\text{ps\_partkey} \wedge \text{ps\_suppkey}=\text{s\_suppkey}} (\text{Part} \times \text{Supplier}))$$

More equivalences for existing optimization techniques

Bellamkonda et al., *Enhanced Subquery Optimizations in Oracle*. In PVLDB, 2009.

Subquery coalescing rule I

$\sigma[(A\ \theta_1\ \theta_3(\pi[B](\sigma[F_1](Inner)))\ \theta_2\ (A\ \theta_1\ \theta_3(\pi[B](\sigma[F_2](Inner)))](Outer)$
$\equiv \sigma[A\ \theta_1\ \theta_3(\pi[B](\sigma[F_1 \vee F_2](Inner))](Outer)$

Subquery coalescing rule II

$\sigma[(A\ \theta_1\ \theta_3(\pi[A](\sigma[F_1](Inner)))\ \theta_2\ (A\ \theta_1\ \theta_3(\pi[A](\sigma[F_2](Inner)))](Outer)$
$\equiv \sigma[A\ \theta_1\ \theta_3(\pi[A](\sigma[F_1](Inner))](Outer)$

Subquery coalescing rule III

$\sigma[(A\ \theta_1\ \theta_3(\pi[A](\sigma[F_1](Inner)))\ \theta_2\ (A\ \theta_1\ \theta_3(\pi[A](\sigma[F_2](Inner)))](Outer)$
$\equiv \sigma[A\ \theta_1\ \theta_3(\pi[A](\sigma[F_2](Inner))](Outer)$

Example: Subquery coalescing

Example

```
SELECT *
FROM Orders
WHERE o_totalprice >= (SELECT MAX(o_totalprice)
                       FROM Orders
                       WHERE o_orderpriority = '2-HIGH')
  AND o_totalprice >= (SELECT MAX(o_totalprice)
                       FROM Orders
                       WHERE o_orderpriority = '3-MEDIUM')
```

$$\Downarrow$$

```
SELECT *
FROM Orders
WHERE o_totalprice >= (SELECT MAX(o_totalprice)
                       FROM Orders
                       WHERE o_orderpriority = '2-HIGH'
                           OR o_orderpriority = '3-MEDIUM')
```

Example: Subquery coalescing

Example

```
SELECT *
FROM Orders
WHERE o_totalprice >= (SELECT MAX(o_totalprice)
                       FROM Orders
                       WHERE o_orderpriority = '2-HIGH')
  AND o_totalprice >= (SELECT MAX(o_totalprice)
                       FROM Orders
                       WHERE o_orderpriority = '3-MEDIUM')
```

Example: Subquery coalescing

Example

```
SELECT *
FROM Orders
WHERE o_totalprice >= (SELECT MAX(o_totalprice)
                       FROM Orders
                       WHERE o_orderpriority = '2-HIGH')
  AND o_totalprice >= (SELECT MAX(o_totalprice)
                       FROM Orders
                       WHERE o_orderpriority = '3-MEDIUM')
```

- NF^2 representation:

Example: Subquery coalescing

Example

```
SELECT *
FROM Orders
WHERE o_totalprice >= (SELECT MAX(o_totalprice)
                       FROM Orders
                       WHERE o_orderpriority = '2-HIGH')
  AND o_totalprice >= (SELECT MAX(o_totalprice)
                       FROM Orders
                       WHERE o_orderpriority = '3-MEDIUM')
```

- NF^2 representation:

Orders

Example: Subquery coalescing

Example

```
SELECT *
FROM Orders
WHERE o_totalprice >= (SELECT MAX(o_totalprice)
                       FROM Orders
                       WHERE o_orderpriority = '2-HIGH')
  AND o_totalprice >= (SELECT MAX(o_totalprice)
                       FROM Orders
                       WHERE o_orderpriority = '3-MEDIUM')
```

- NF^2 representation:

$$\sigma[\text{o\_totalprice} \geq \text{MAX}(\pi[\text{o\_totalprice'}](\sigma[\text{o\_orderpriority} = \text{'2-HIGH'}](\text{Orders'}))) \quad](\text{Orders})$$

Example: Subquery coalescing

Example

```
SELECT *
FROM Orders
WHERE o_totalprice >= (SELECT MAX(o_totalprice)
                       FROM Orders
                       WHERE o_orderpriority = '2-HIGH')
  AND o_totalprice >= (SELECT MAX(o_totalprice)
                       FROM Orders
                       WHERE o_orderpriority = '3-MEDIUM')
```

- NF^2 representation:

$$\sigma[\text{o\_totalprice} \geq \text{MAX}(\pi[\text{o\_totalprice'}](\sigma[\text{o\_orderpriority} = \text{'2-HIGH'}](\text{Orders'}))) \wedge \text{o\_totalprice} \geq \text{MAX}(\pi[\text{o\_totalprice'}](\sigma[\text{o\_orderpriority} = \text{'3-MEDIUM'}](\text{Orders'})))](\text{Orders})$$

Example: Subquery coalescing

Equivalence rule

$$\begin{aligned}
&\sigma[A \geq \text{MAX}(\pi[B](\sigma[F_1](Inner))) \wedge \\
&\quad A \geq \text{MAX}(\pi[B](\sigma[F_2](Inner)))](Outer) \\
&\equiv \sigma[A \geq \text{MAX}(\pi[B](\sigma[F_1 \vee F_2](Inner))](Outer)
\end{aligned}$$

Example

σ[o_totalprice $\geq$ MAX(π[o_totalprice'](
σ[o_orderpriority = '2-HIGH'](Orders'))) $\wedge$
o_totalprice $\geq$ MAX(π[o_totalprice'](
σ[o_orderpriority = '3-MEDIUM'](Orders')))](Orders)

Example: Subquery coalescing

Equivalence rule

$$\sigma[A \geq \text{MAX}(\pi[B](\sigma[F_1](Inner))) \wedge$$
$$A \geq \text{MAX}(\pi[B](\sigma[F_2](Inner)))](Outer)$$
$$\equiv \sigma[A \geq \text{MAX}(\pi[B](\sigma[F_1 \vee F_2](Inner))](Outer)$$

Example

$$\sigma[\text{o\_totalprice} \geq \text{MAX}(\pi[\text{o\_totalprice'}]($$
$$\sigma[\text{o\_orderpriority} = \text{'2-HIGH'}](\text{Orders'}))) \wedge$$
$$\text{o\_totalprice} \geq \text{MAX}(\pi[\text{o\_totalprice'}]($$
$$\sigma[\text{o\_orderpriority} = \text{'3-MEDIUM'}](\text{Orders'})))](\text{Orders})$$

Example: Subquery coalescing

Equivalence rule

$$\sigma[A \geq \text{MAX}(\pi[B](\sigma[F_1](Inner))) \wedge$$
$$A \geq \text{MAX}(\pi[B](\sigma[F_2](Inner)))](Outer)$$
$$\equiv \sigma[A \geq \text{MAX}(\pi[B](\sigma[F_1 \vee F_2](Inner))](Outer)$$

Example

σ[o_totalprice $\geq$ MAX(π[o_totalprice'](
σ[o_orderpriority = '2-HIGH'](Orders'))) $\wedge$
o_totalprice $\geq$ MAX(π[o_totalprice'](
σ[o_orderpriority = '3-MEDIUM'](Orders')))](Orders)

Example: Subquery coalescing

Equivalence rule

$$\sigma[A \geq \mathrm{MAX}(\pi[B](\sigma[F_1](Inner))) \wedge$$
$$A \geq \mathrm{MAX}(\pi[B](\sigma[F_2](Inner)))](Outer)$$
$$\equiv \sigma[A \geq \mathrm{MAX}(\pi[B](\sigma[F_1 \vee F_2](Inner))](Outer)$$

Example

σ[o_totalprice $\geq$ MAX(π[o_totalprice'](
σ[o_orderpriority $=$ '2-HIGH'](Orders'))) $\wedge$
o_totalprice $\geq$ MAX(π[o_totalprice'](
σ[o_orderpriority $=$ '3-MEDIUM'](Orders'))))](Orders)

$\equiv$ σ[o_totalprice $\geq$ MAX(π[o_totalprice'](
σ[o_orderpriority' $=$ '2-HIGH' $\vee$
o_orderpriority' $=$ '3-MEDIUM'](Orders'))))](Orders)

Example: Subquery coalescing

Equivalence rule

$$\sigma[A \geq \text{MAX}(\pi[B](\sigma[F_1](Inner))) \wedge$$
$$A \geq \text{MAX}(\pi[B](\sigma[F_2](Inner)))](Outer)$$
$$\equiv \sigma[A \geq \text{MAX}(\pi[B](\sigma[F_1 \vee F_2](Inner))](Outer)$$

Example

$$\sigma[\text{o\_totalprice} \geq \text{MAX}(\pi[\text{o\_totalprice'}]($$
$$\sigma[\text{o\_orderpriority} = \text{'2-HIGH'}](\text{Orders'}))) \wedge$$
$$\text{o\_totalprice} \geq \text{MAX}(\pi[\text{o\_totalprice'}]($$
$$\sigma[\text{o\_orderpriority} = \text{'3-MEDIUM'}](\text{Orders'})))](\text{Orders})$$

$$\equiv \sigma[\text{o\_totalprice} \geq \text{MAX}(\pi[\text{o\_totalprice'}]($$
$$\sigma[\text{o\_orderpriority'} = \text{'2-HIGH'} \vee$$
$$\text{o\_orderpriority'} = \text{'3-MEDIUM'}](\text{Orders'})))](\text{Orders})$$

Example: Subquery coalescing

Equivalence rule

$$\sigma[A \geq \mathrm{MAX}(\pi[B](\sigma[F_1](Inner))) \wedge$$
$$A \geq \mathrm{MAX}(\pi[B](\sigma[F_2](Inner)))](Outer)$$
$$\equiv \sigma[A \geq \mathrm{MAX}(\pi[B](\sigma[F_1 \vee F_2](Inner))](Outer)$$

Example

$$\sigma[\text{o\_totalprice} \geq \mathrm{MAX}(\pi[\text{o\_totalprice'}]($$
$$\sigma[\text{o\_orderpriority} = \text{'2-HIGH'}](\text{Orders'}))) \wedge$$
$$\text{o\_totalprice} \geq \mathrm{MAX}(\pi[\text{o\_totalprice'}]($$
$$\sigma[\text{o\_orderpriority} = \text{'3-MEDIUM'}](\text{Orders'})))](\text{Orders})$$

$$\equiv \sigma[\text{o\_totalprice} \geq \mathrm{MAX}(\pi[\text{o\_totalprice'}]($$
$$\sigma[\text{o\_orderpriority'} = \text{'2-HIGH'} \vee$$
$$\text{o\_orderpriority'} = \text{'3-MEDIUM'}](\text{Orders'})))](\text{Orders})$$

Evaluation

SQL

$\downarrow$

NF^2

$\downarrow$

NF^2 optimizer

$\downarrow$

Execution plan

$\downarrow$

SQL

$\downarrow$

Execution in DB

Evaluation

SQL

↓

NF^2

↓

NF^2 optimizer

↓

Execution plan

↓

SQL

↓

Execution in DB

Set of 11 nested queries

- Subqueries in SELECT, FROM and WHERE clause
- Subqueries with multiple nestings
- Subqueries with redundancy

Evaluation

SQL

$\downarrow$

NF^2

$\downarrow$

NF^2 optimizer

$\downarrow$

Execution plan

$\downarrow$

SQL

$\downarrow$

Execution in DB

- 1:1 translation from SQL

Evaluation

SQL

$\downarrow$

NF^2

$\downarrow$

NF^2 optimizer

$\downarrow$

Execution plan

$\downarrow$

SQL

$\downarrow$

Execution in DB

- Optimize each query **with** and **without** NF^2 rules

Evaluation

SQL

$\downarrow$

NF^2

$\downarrow$

NF^2 optimizer

$\downarrow$

Execution plan

$\downarrow$

SQL

$\downarrow$

Execution in DB

- Generated by the NF^2 optimizer

Evaluation

SQL

$\downarrow$

NF^2

$\downarrow$

NF^2 optimizer

$\downarrow$

Execution plan

$\downarrow$

SQL

$\downarrow$

Execution in DB

- Derived from execution plan

Evaluation

SQL

$\downarrow$

NF^2

$\downarrow$

NF^2 optimizer

$\downarrow$

Execution plan

$\downarrow$

SQL

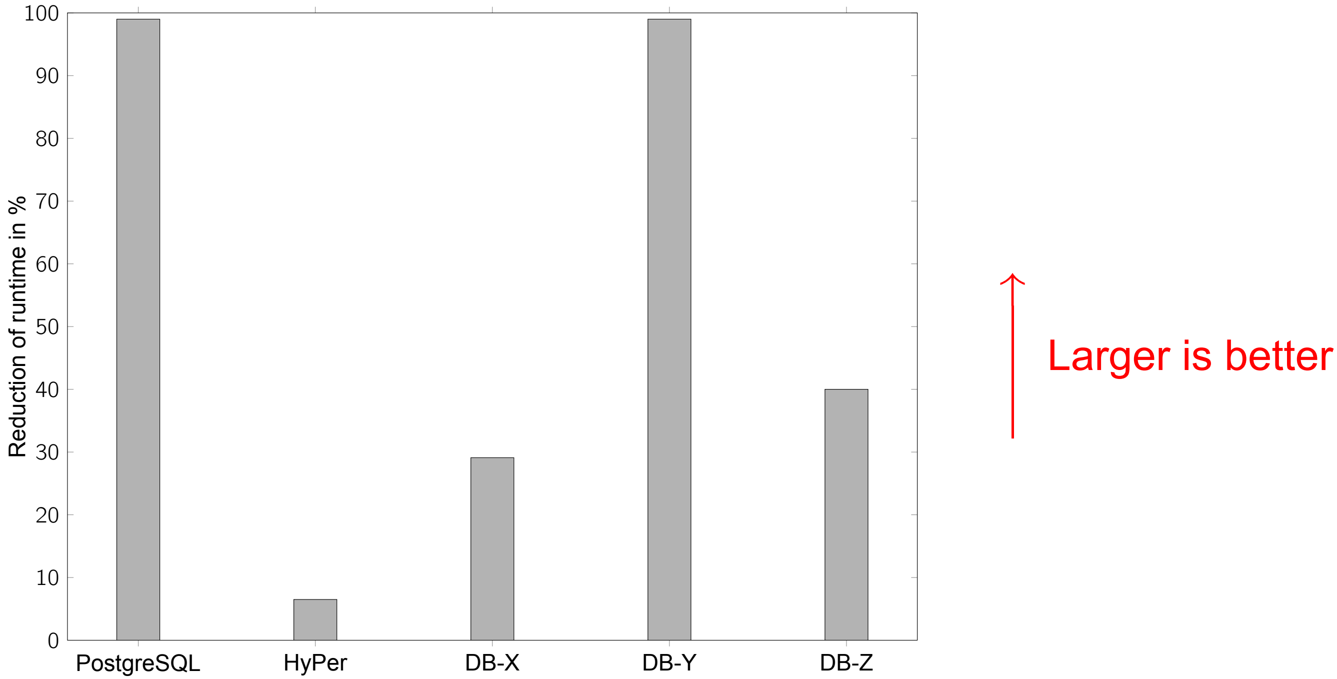
$\downarrow$

Execution in DB

Systems:

- Postgres 9.4.2
- HyPer
- Three commercial database systems

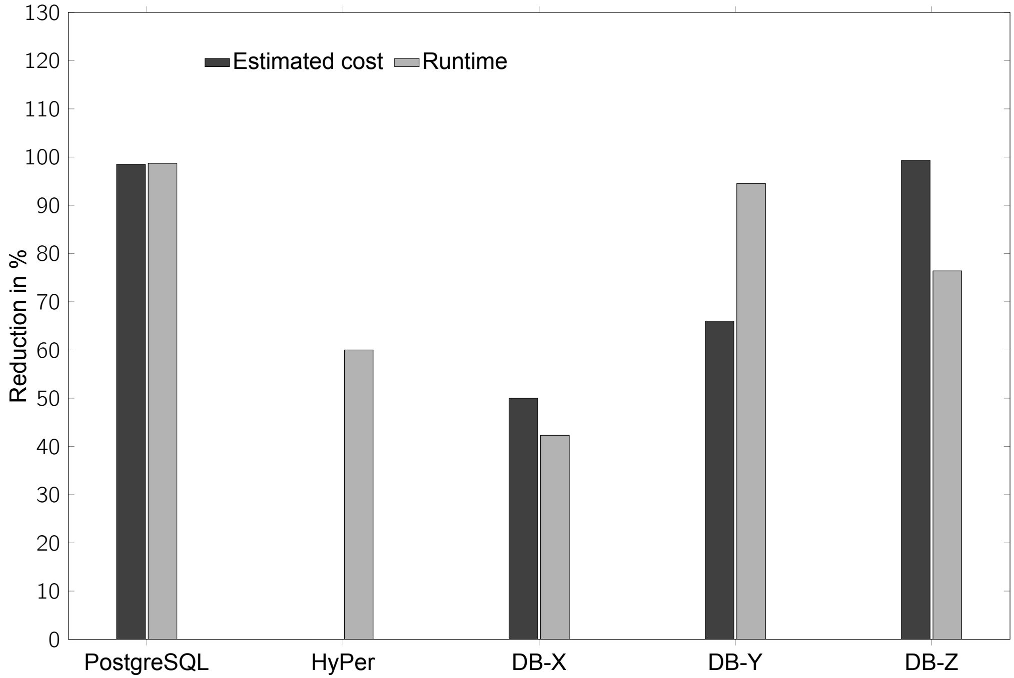
Evaluation: Runtime reduction over all queries



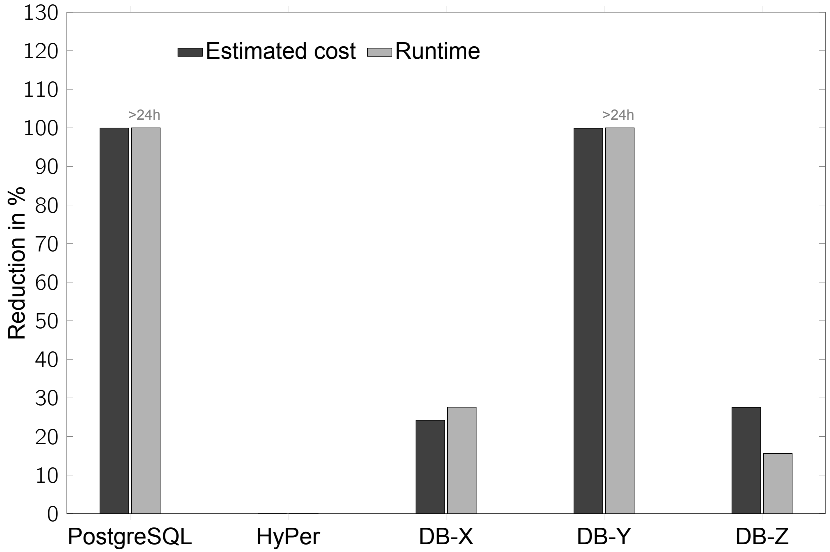
Formula to compute the runtime reduction:

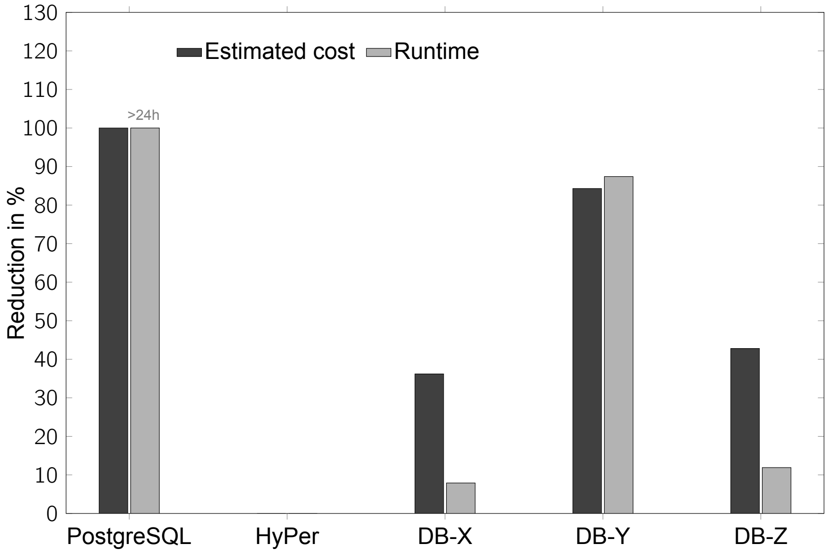
$$\frac{(\text{runtime original query} - \text{runtime transformed query})}{\text{runtime original query}} * 100$$

Evaluation: Subquery in Select Clause

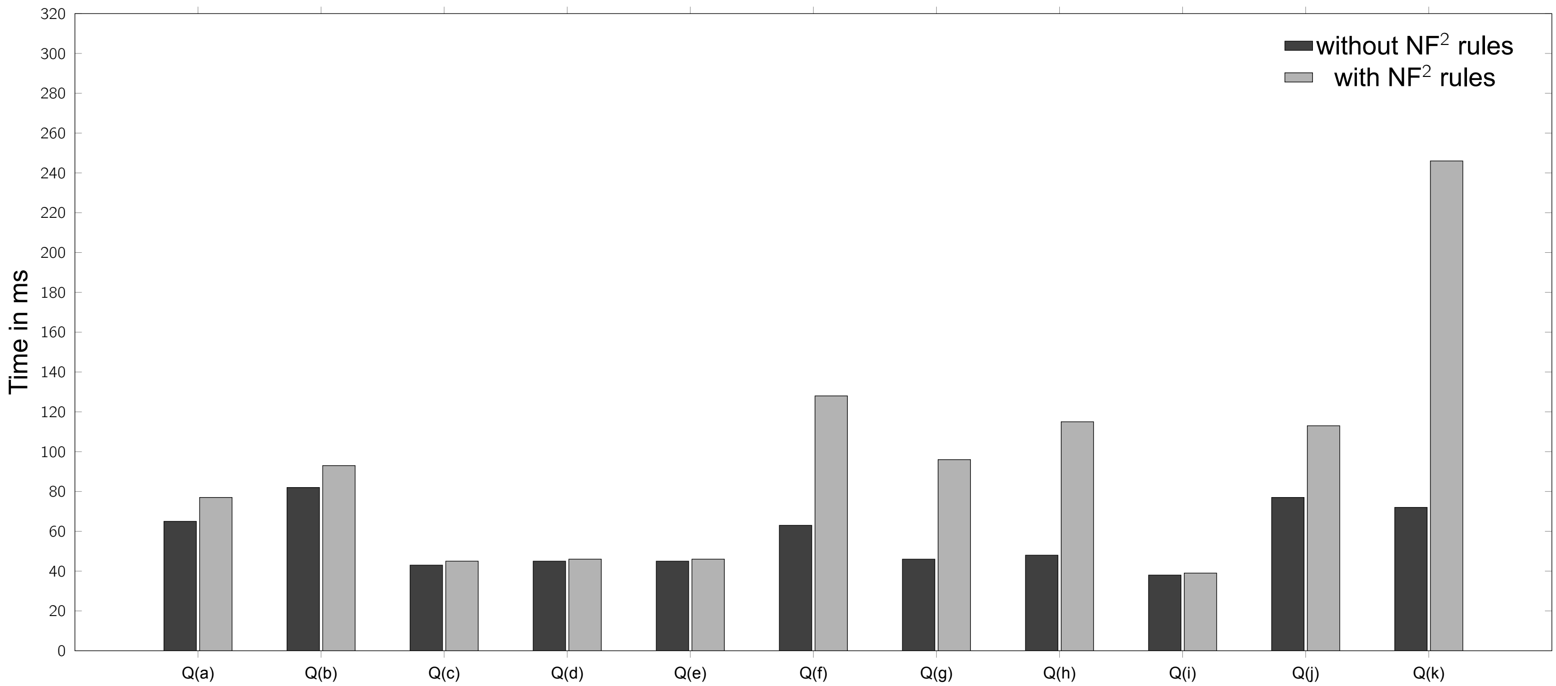


Evaluation: Subqueries with Multiple Nestings

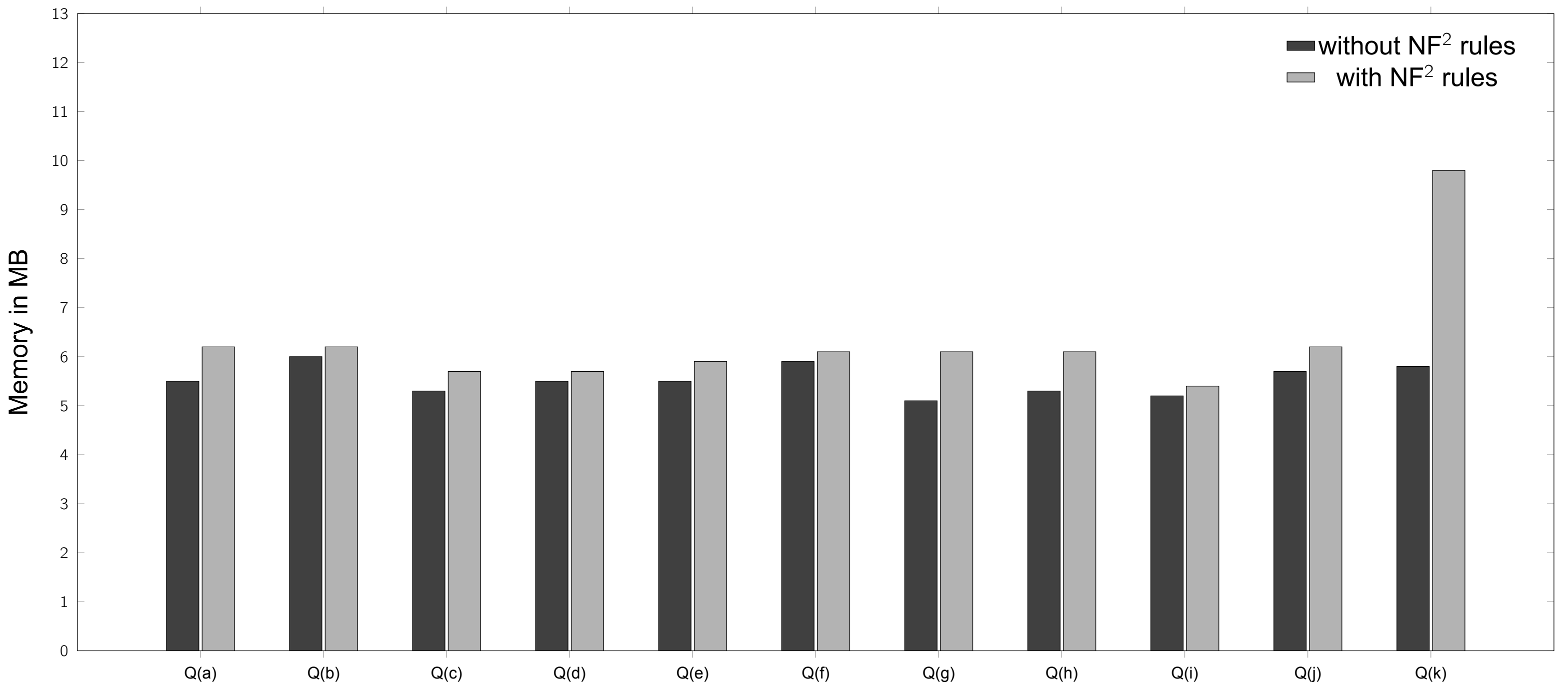




Evaluation: NF^2 Optimizer (Runtime)



Evaluation: NF^2 Optimizer (Memory)



Summary

NF^2 algebra for optimizing nested queries

Summary

NF^2 algebra for optimizing nested queries

- Can represent all types of nested queries

Summary

NF^2 algebra for optimizing nested queries

- Can represent all types of nested queries
- Extension of the relational algebra
  $\Rightarrow$ Existing equivalences remain valid

Summary

NF^2 algebra for optimizing nested queries

- Can represent all types of nested queries
- Extension of the relational algebra
  $\Rightarrow$ Existing equivalences remain valid
- Can be integrated into a transformation-based optimizer with little effort

Summary

NF^2 algebra for optimizing nested queries

- Can represent all types of nested queries
- Extension of the relational algebra
  $\Rightarrow$ Existing equivalences remain valid
- Can be integrated into a transformation-based optimizer with little effort
- No NF^2 backend is needed