Overview

- rewrite query into semantically equivalent but more efficientl form
- same query can be expressed differently
  - avoid hand-tuning
- complex queries may have redundancy
- e.g. remove distinct
  - primary key is always unique
  - no need to check whether it already exists
    - **A Simple Example**
      - Catalog meta data: custkey is unique

```
SELECT DISTINCT custkey, name
  FROM TPCH.Customer
```

rewrite

```
SELECT custkey, name
  FROM TPCH.Customer
```

  –

Standardization and Simplification

- **Normal Forms of Boolean Expressions**
  - **Conjunctive** normal form (P_{11} OR ... OR P_{1n}) **AND ... AND** (P_{m1} OR ... OR P_{mp})
  - **Disjunctive** normal form (P_{11} AND ... AND P_{1q}) **OR ... OR** (P_{r1} AND ... AND P_{rs})
- **Transformation Rules for Boolean Expressions**

Rule Name	Examples	
Commutativity rules	A OR B ⇔ B OR A A AND B ⇔ B AND A	
Associativity rules	(A OR B) OR C ⇔ A OR (B OR C) (A AND B) AND C ⇔ A AND (B AND C)	
Distributivity rules	A OR (B AND C) ⇔ (A OR B) AND (A OR C) A AND (B OR C) ⇔ (A AND B) OR (A AND C)	
De Morgan's rules	NOT (A AND B) ⇔ NOT (A) OR NOT (B) NOT (A OR B) ⇔ NOT (A) AND NOT (B)	
Double-negation rules	NOT(NOT(A)) ⇔ A	
Idempotence rules	A OR A ⇔ A A OR NOT(A) ⇔ TRUE A AND (A OR B) ⇔ A A OR FALSE ⇔ A A AND FALSE ⇔ FALSE	A AND A ⇔ A A AND NOT (A) ⇔ FALSE A OR (A AND B) ⇔ A A AND TRUE ⇔ A A OR TRUE ⇔ TRUE

- **Elimination of Common Subexpressions**
  - $(A_1{=}a_{11}$ **OR** $A_1{=}a_{12})$ **AND** $(A_1{=}a_{12}$ **OR** $A_1{=}a_{11}) \rightarrow A_1{=}a_{11}$ **OR** $A_1{=}a_{12}$
- **Propagation of Constants**
  - A ≥ B **AND** B = 7 → A ≥ 7 **AND** B = 7
  - $R \bowtie_{a=b}(\sigma_{b>0}(S)) \rightarrow (\sigma_{a>0}(R)) \bowtie_{a=b}(\sigma_{b>0}(S))$
- **Detection of Contradictions**
  - A ≥ B **AND** B > C **AND** C ≥ A → **A > A** → **FALSE**
- **Use of Constraints**
  - A is primary key/unique: π_A → no duplicate elimination necessary
  - Rule MAR_STATUS = 'married' → TAX_CLASS ≥ 3:
    (MAR_STATUS = 'married' **AND** TAX_CLASS = 1) → **FALSE**
- **Elimination of Redundancy** (set semantics)
  - $R \bowtie R \rightarrow R$, $R \cup R \rightarrow R$, $R - R \rightarrow \emptyset$
  - $R \bowtie (\sigma_p R) \rightarrow \sigma_p R$, $R \cup (\sigma_p R) \rightarrow R$, $R - (\sigma_p R) \rightarrow \sigma_{\neg p} R$
  - $(\sigma_{p1} R) \bowtie (\sigma_{p2} R) \rightarrow \sigma_{p1 \wedge p2} R$, $(\sigma_{p1} R) \cup (\sigma_{p2} R) \rightarrow \sigma_{p1 \vee p2} R$

Query Unnesting

- type-A nesting
  - unrelated inner query computes an aggregate
  - no need to aggregate for each tuple
  - instead aggregate once and insert result into outer query

```
SELECT OrderNo FROM Order
  WHERE ProdNo =
   (SELECT MAX(ProdNo)
      FROM Product WHERE Price<100)
```

```
$X = SELECT MAX(ProdNo)
  FROM Product WHERE Price<100
SELECT OrderNo FROM Order
  WHERE ProdNo = $X
```

- type-N nesting
  - unrelated inner query, which returns set of tuples
  - join more efficient

```
SELECT OrderNo FROM Order
  WHERE ProdNo IN
   (SELECT ProdNo
      FROM Product WHERE Price<100)
```

```
SELECT OrderNo
 FROM Order O, Product P
 WHERE O.ProdNo = P.ProdNo
   AND P.Price < 100
```

- type-J nesting
  - unnesting of correlated subqueries w/o aggregation
  - optimized via join constraint
  - instead of constraint within subqery

```
SELECT OrderNo FROM Order O
  WHERE ProdNo IN
   (SELECT ProdNo FROM Project P
    WHERE P.ProjNo = O.OrderNo
      AND P.Budget > 100,000)
```

```
SELECT OrderNo
  FROM Order O, Project P
  WHERE O.ProdNo = P.ProdNo
    AND P.ProjNo = O.OrderNo
    AND P.Budget > 100,000
```

- type-JA nesting
  - unnesting of correlated subqueries w/ aggregation

– all aggregates computed at once

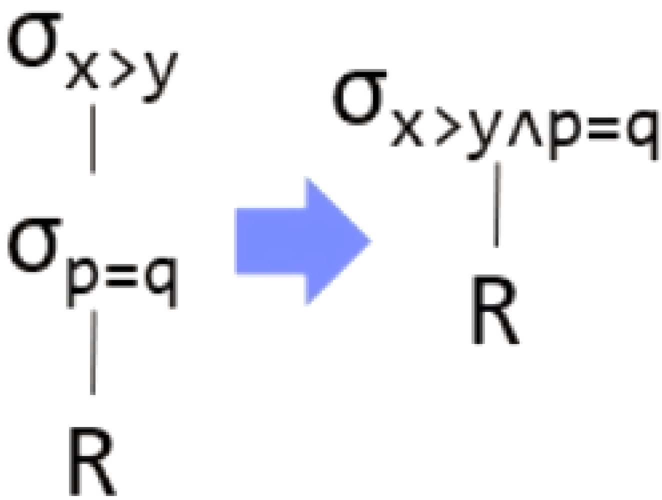
```sql
SELECT OrderNo FROM Order O
  WHERE ProdNo IN
   (SELECT MAX(ProdNo)
     FROM Project P
     WHERE P.ProjNo = O.OrderNo
       AND P.Budget > 100,000)
```

```sql
SELECT OrderNo FROM Order O
  WHERE ProdNo IN
   (SELECT ProdNo FROM
     (SELECT ProjNo, MAX(ProdNo)
       FROM Project
       WHERE Budget > 100.000
       GROUP BY ProjNo) P
     WHERE P.ProjNo = O.OrderNo)
```

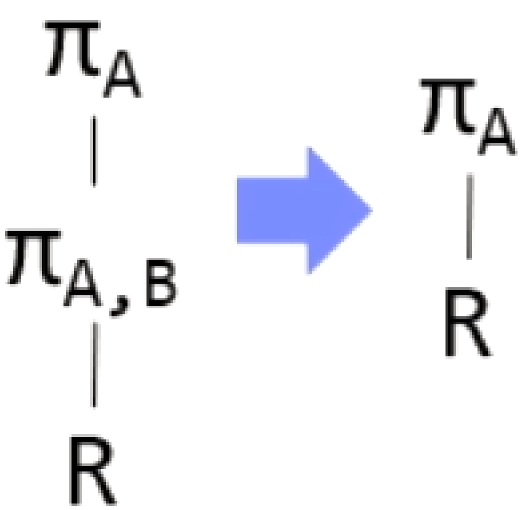
- Further un-nesting via case 3 and 2

Selections and Projections

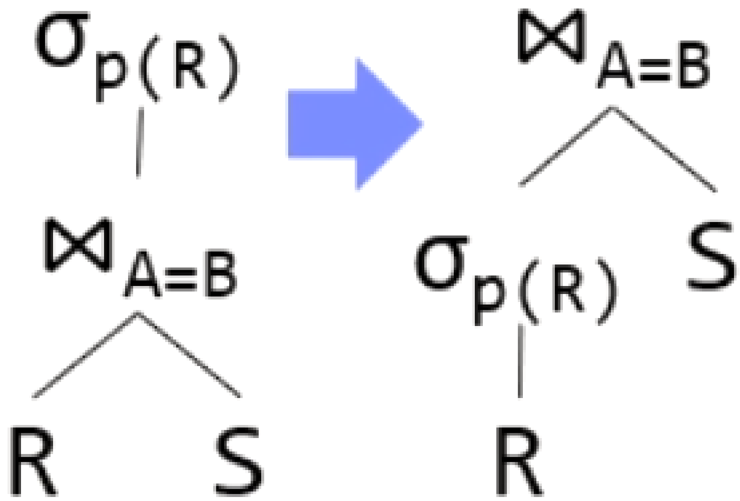
- transformation rules
  - selection grouping
    - multiple groups combined to one



  - projection grouping
    - instead of filtering into stricter filtering
    - only stricter filtering

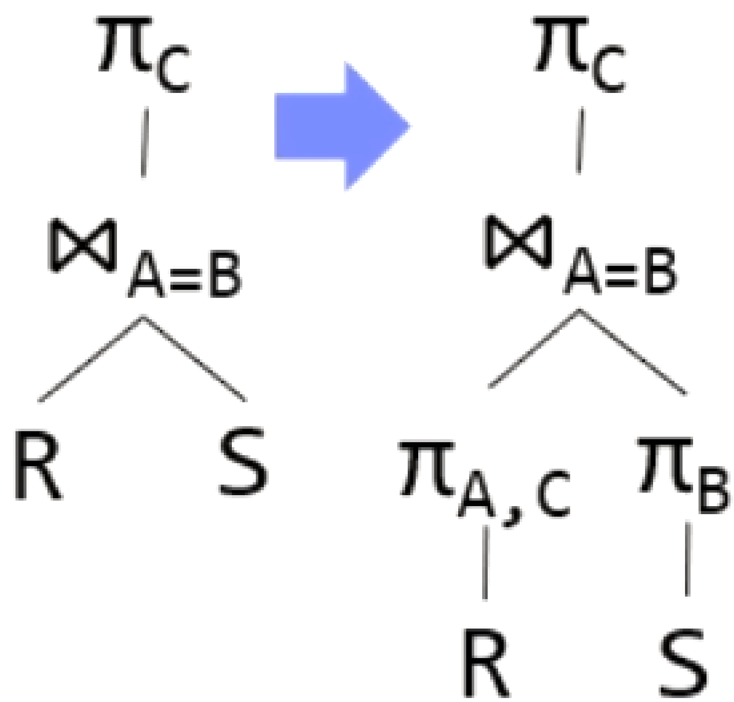


- selection pushdown
  * allows moving selection after join to before
  * reduces size of join inputs
    - may allow storing all data within RAM



  *
- projection pushdown
  * if only some joined columns are required
  * remove other columns before

4) Pushdown of Projections



*

- restructuring algorith

#1 Split n-ary joins into binary joins

#2 Split multi-term selections

#3 Push-down selections as far as possible

#4 Group adjacent selections again

#5 Push-down projections as far as possible

–

Input: Standardized, simplified, and un-nested query graph

Output: Restructured query graph

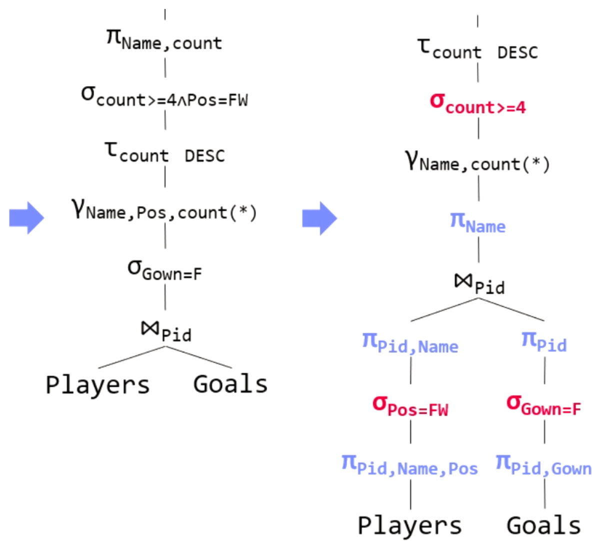
–

- examples

```
SELECT Name, count
  FROM TopScorer
  WHERE count>=4
    AND Pos='FW'

CREATE VIEW TopScorer AS
SELECT P.Name, P.Pos, count(*)
  FROM Players P, Goals G
  WHERE P.Pid=G.Pid
    AND G.GOwn=FALSE
  GROUP BY P.Name, P.Pos
  ORDER BY count(*) DESC
```

Additional metadata:
P.Name is unique



–

- $\sigma_{b=7}(R \bowtie S)$ → $\sigma_{b=7}(R) \bowtie S$
- $(\sigma_{e>3}(S)) \cap (\sigma_{f<7}(S))$ → $\sigma_{e>3 \wedge f<7}(S)$
- $\pi_{a,b}(R \bowtie_{a=d} S)$ → $\pi_{a,b}(R) \ltimes_{a=d} S$
- $R \cup (\sigma_{d<e \wedge e<f \wedge f<d}(S))$ → $R \cup \emptyset$ → R
- $\sigma_{b=3}(\gamma_{b,max(c)}(R))$ → $\gamma_{3,max(c)}(\sigma_{b=3}(R))$

–

Expression 1	Expression 2	Equivalent?
$\sigma_{c=3}(\sigma_{b=7}(R))$	$\sigma_{c=3}(\sigma_{c=3 \vee b=7}(R))$	✗
$R \bowtie_{a=e} S$	$\sigma_{a=e}(R \times S)$	✓
$(\sigma_{b<3}(R)) \cap (\sigma_{b \geq 3}(R))$	R	✗
$\pi_{b,d}(R \bowtie_{a=e} S)$	$(\pi_{a,b}(R)) \bowtie_{a=e} (\pi_{d,e}(S))$	✗
$\pi_{a,b}(\sigma_{c=3}(\sigma_{b=7}(R)))$	$\sigma_{b=7}(\pi_{a,b}(\sigma_{c=3}(R)))$	✓

–