CS5226 Lecture 2
Query Tuning

Query Tuning

- Eliminating redundant DISTINCT
- Rewriting nested queries

Review: Relational Algebra

- $\pi_S R$ = projection of relation R onto the set of columns in S
- $\pi_S^{all} R$ = projection of relation R onto the set of columns in S (duplicates preserved)
- $\sigma_p R$ = selection of relation R
  - p = predicate to filter qualifying rows from R
- $\uplus$ = union with duplicates preserved (SQL's union all)
- $columns(X)$ = set of columns in X, where X is a relation or predicate

Review: Relational Algebra (cont.)

- $R \times S$ = cross product
- $R \bowtie_p S$ = inner-join
- $R \ltimes_p S$ = left semi-join
- $R \rtimes_p S$ = right semi-join
- $R \triangleright_p S$ = anti-join
- $R ⟕_p S$ = left outer-join (LOJ)
- $R ⟖_p S$ = right outer-join (ROJ)
- $R ⟗_p S$ = full outer-join (FOJ)

Review: Functional Dependencies

- **Functional dependencies (FDs)** are constraints on schemas that specify that the values for a certain set of attributes determine unique values for another set of attributes
- Let α and β denote subsets of attributes of a relational schema *R* (i.e., $\alpha, \beta \subseteq$ *columns*(*R*))
- We use $\alpha \rightarrow \beta$ to denote that α functionally determines β (or β functionally depends on α)

Review: Functional Dependencies (cont.)

- Let $\pi_\alpha(t)$ denote the projection of α on tuple t
- Let r be a relation instance of relation schema R
- r satisfies FD $\alpha \rightarrow \beta$ if for every pair of tuples t_1 and t_2 in r such that $\pi_\alpha(t_1) = \pi_\alpha(t_2)$, it is also true that $\pi_\beta(t_1) = \pi_\beta(t_2)$
- A FD $\alpha \rightarrow \beta$ holds on R iff for any relation instance r of R, r satisfies $\alpha \rightarrow \beta$

Review: Functional Dependencies (cont.)

- Consider the following relation instance r:

Module	**Prof**	**Room**	**Building**	**Time**
CS101	Turing	LT 1	CS	0800
CS400	Turing	LT 1	CS	1400
MU300	Bach	LT 2	Math	1400
MA200	Newton	LT 2	Math	1000
CS101	Turing	LT 2	Math	1200

- r satisfies Room $\rightarrow$ Building
- r does not satisfy Prof $\rightarrow$ Module

Review: Functional Dependencies (cont.)

- A set of attributes α is a superkey of schema *R* if for every instance *r* of *R*, *r* has no duplicate values for α
- α is a superkey of *R* iff $\alpha \rightarrow \mathit{columns}(R)$
- α is a key of *R* if α is a superkey of *R* and no proper subset of α is a superkey of *R*

Review: Reasoning about FDs

- Let F be a set of FDs and f be an FD
- F logically implies (or implies) f if every relation instance of R that satisfies the FDs F also satisfies the FD f

Inference Rules for FDs

- Let $\alpha, \beta, \gamma \subseteq$ *columns*(R)
- Reflexivity: If $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
- Augmentation: If $\alpha \rightarrow \beta$, then $\alpha\gamma \rightarrow \beta\gamma$
- Transitivity: If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$
- Union: If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$, then $\alpha \rightarrow \beta\gamma$
- Decomposition: If $\alpha \rightarrow \beta$, then $\alpha \rightarrow \beta'$ for any $\beta' \subseteq \beta$

Review: Reasoning about FDs (cont.)

- Let $\alpha \subseteq \mathit{columns}(R)$
- Let F be a set of FDs that hold on R
- The closure of α (with respect to F), denoted by α^+, is the set of attributes that are functionally determined by α with respect to F

$$\alpha^+ = \{A \in \mathit{columns}(R) \mid F \text{ implies } \alpha \rightarrow A\}$$

- F implies $\alpha \rightarrow \beta$ if and only if $\beta \subseteq \alpha^+$ (w.r.t. F)

Review: Reasoning about FDs (cont.)

Input: α, *F*
Output: α^+ (w.r.t. *F*)

$\alpha^+ = \alpha$
stop = false
repeat
    if (there exists some FD $\beta \to \gamma \in F$ such that $\beta \subseteq \alpha^+$ and $\gamma \not\subseteq \alpha^+$) then
        $\alpha^+ = \alpha^+ \cup \gamma$
    else
        stop = true
until (stop)
return α^+

Eliminating Redundant DISTINCT

- Customer (cust#, cname, country)
- SalesRep (rep#, sname, country)
- SalesOffice (country, address, phone)

Q1:
```
select distinct cust#
from   Customer, SalesRep
where  Customer.country = SalesRep.country
```

Q2:
```
select distinct cust#
from   Customer, SalesOffice
where  Customer.country = SalesOffice.country
```

Eliminating Redundant DISTINCT

- **R**(A,B,C)
- **S**(D,E,F)

1. **select distinct** B **from** R
2. **select distinct** A, B **from** R
3. **select distinct** A, D, E **from** R, S
4. **select distinct** A, E **from** R, S
5. **select distinct** A, E **from** R, S **where** A = F
6. **select distinct** A, E **from** R, S **where** B = D

SPJ Queries

A SPJ query is a query of the form

$$\pi_S^{all}\sigma_p(R_1 \times \cdots \times R_n)$$

- $S \subseteq \bigcup_{i=1}^{n} columns(R_i)$
- $p = p_1 \wedge \cdots \wedge p_m$
  where each p_i is of the form "A op c" or "A op A'"
  - op is a comparison operator
  - c is a constant
  - $A, A' \in \bigcup_{i=1}^{n} columns(R_i)$

Eliminating Redundant DISTINCT

- Let $Q = \pi_S^{all} \sigma_p(R_1 \times \cdots \times R_n)$ be a SPJ query
- Let F be the set of FDs defined as follows:
  - For each FD $X \rightarrow Y$ that holds on $R_1, \cdots, R_n$, $X \rightarrow Y \in F$
  - For each selection predicate $A = c$ in p,

    $$\{A' \rightarrow A \mid A' \in \bigcup_{i=1}^{n} columns(R_i)\} \subseteq F$$
  - For each equality join predicate $A = A'$ in p, $\{A \rightarrow A', A' \rightarrow A\} \subseteq F$

If S is a superkey of $\sigma_p(R_1 \times \cdots \times R_n)$ wrt F, then the result of Q has no duplicates

Example

- **R**(A,B,C)
- **S**(D,E,F)
- **T**(G,H)

select **distinct** B, D
from R, S, T
where A = E
and C = H
and G = 10

Rewriting of Nested Queries

- Customer (cust#, cname, country)
- Order (order#, cust#, date, totalprice)

```
select cname
from   Customer c
where  1000 <
       (select sum(o.totalprice)
        from   Order o
        where  o.cust# = c.cust#)
```

```
select   cname
from     Customer c join Order o
         on c.cust# = o.cust#
group by c.cust#, c.cname
having   1000 < sum(o.totalprice)
```

Nested Queries

- A nested query is a query containing some subquery
- A subquery in a nested query is also called an inner query which is contained in an outer query
- A correlated nested query is a nested query where there is a subquery that is dependent on the tuple referenced in its outer query

Subqueries in SQL

- Used as a scalar expression in SELECT clause

```
select order#,
       (select c.cname
        from   Customer c
        where  c.cust# = o.cust#)
from   Order o
```

Subqueries in SQL (cont.)

- Used as a derived table in FROM clause

```
select c.cname
from   Customer c,
       (select   cust#
        from     Order
        group by cust#
        having   1000 < sum(totalprice)
       ) as o
where  c.cust# = o.cust#
```

Subqueries in SQL (cont.)

- Used as a scalar expression in WHERE / HAVING clause

```
select  cname
from    Customer c
where   1000 <
        (select sum(o.totalprice)
         from   Order o
         where  o.cust# = c.cust#)
```

Classification of nested queries

- Is the nested query correlated or uncorrelated?
- Does the subquery involve aggregation?

Classification of nested queries (cont.)

Q1:

```
select  order#
from    Order
where   cust# in (select cust#
                  from Customer
                  where country = "Singapore")
```

Q2:

```
select  order#
from    Order
where   totalprice = (select max(totalprice) from Order)
```

Classification of nested queries (cont.)

Q3:

```
select cname
from   Customer c
where  exists (select *
               from   Order o
               where  o.cust# = c.cust#
               and    totalprice > 5000)
```

Q4:

```
select cname
from   Customer c
where  1000 <
       (select sum(o.totalprice)
        from   Order o
        where  o.cust# = c.cust#)
```

How to unnest subqueries?

```
select  cname
from    Customer c
where   1000 <
        (select sum(o.totalprice)
         from   Order o
         where  o.cust# = c.cust#)
```

```
select    cname
from      Customer c join Order o
          on c.cust# = o.cust#
group by  c.cust#, c.cname
having    1000 < sum(o.totalprice)
```

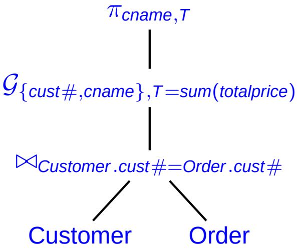
- Transform subqueries into relational algebra expressions
  - Using a new relational operator: apply operator
- Apply rewriting rules to eliminate apply operator

More Relational Algebra

$\mathcal{G}_{A,F}\ R$ = group by on relation R

- A = set of grouping columns
- F = set of aggregate functions

```
select     cname, sum(totalprice) as T
from       Customer c join Order o
           on c.cust# = o.cust#
group by cust#, cname
```

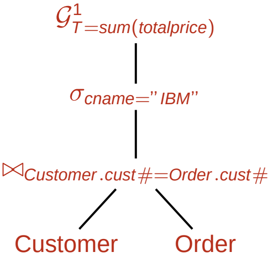


More Relational Algebra (cont.)

$\mathcal{G}^1_F\ R$ = scalar aggregation on relation R

- F = set of aggregate functions
- $\mathcal{G}^1_F\ R$ returns exactly one row

```
select  order#
from    Order
where   totalprice >
        (select sum(totalprice)
        from    Customer c join Order o
                on c.cust# = o.cust#
        where   cname = "IBM"
        )
```



More Relational Algebra (cont.)

- $\mathcal{G}_{A,F}\ \emptyset = \emptyset$
- $\mathcal{G}^1_F\ R$ returns exactly one row
- Thus, $\mathcal{G}_{\emptyset,F}\ R \neq \mathcal{G}^1_F\ R$

$$\mathcal{G}^1_F\ R = \begin{cases} \mathcal{G}_{\emptyset,F}\ R & \text{if } R \neq \emptyset \\ (F_1(\emptyset), \cdots, F_n(\emptyset)) & \text{otherwise} \end{cases}$$

where $F = \{F_1, \cdots, F_n\}$

More Relational Algebra (cont.)

- If *R* is an empty relation, then

Query	Result
SELECT COUNT(A) FROM R	0
SELECT COUNT(*) FROM R	0
SELECT MIN(A) FROM R	NULL
SELECT MAX(A) FROM R	NULL
SELECT SUM(A) FROM R	NULL
SELECT AVG(A) FROM R	NULL

Parameterized Relational Expression

```
select cname
from   Customer c
where  1000 <
       (select sum(o.totalprice)
        from   Order o
        where  o.cust# = c.cust#)
```

E(c) = PRE with parameter $c \in \mathit{Customer}$

```
select cname
from   Customer c
where  1000 < E(c)
```

Parameterized Relational Expression

select cname
from Customer c
where 1000 $<$
(select sum(o.totalprice)
from Order o
where o.cust# = c.cust#)

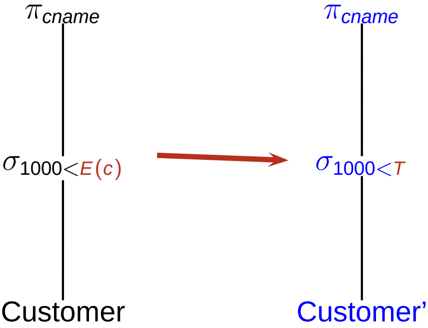
$$\pi_{cname}$$
$$\sigma_{1000 < E(c)}$$
Customer

Query

$$\mathcal{G}^{1}_{T=sum(totalprice)}$$
$$\sigma_{cust\# = c.cust\#}$$
Order

E(c)

Eliminating PRE



- Customer' = Customer + additional column *T*
  - $T = E(c)$
- $Customer' = \biguplus_{c \in Customer} (\{c\} \times E(c))$

Eliminating PRE (cont.)

- $Customer' = \biguplus_{c \in Customer} (\{c\} \times E(c))$
- $E(c) = \mathcal{G}^1_{T=sum(totalprice)}(\sigma_{cust\#=c.cust\#}(Order))$

Customer

cust#	cname	country
1	Alice	...
2	Bob	...
3	Carol	...

Order

order#	cust#	date	totalprice
100	2	...	3000
201	2	...	5000
460	3	...	4000
500	3	...	1000

Customer'

cust#	cname	country	T
1	Alice	...	null
2	Bob	...	8000
3	Carol	...	5000

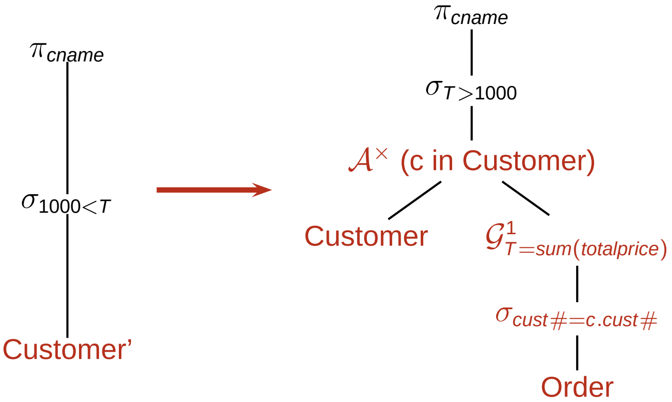
Apply Operator $\mathcal{A}^{\otimes}$

$$R \; \mathcal{A}^{\otimes} \; E \; = \biguplus_{r \in R} (\{r\} \otimes E(r))$$

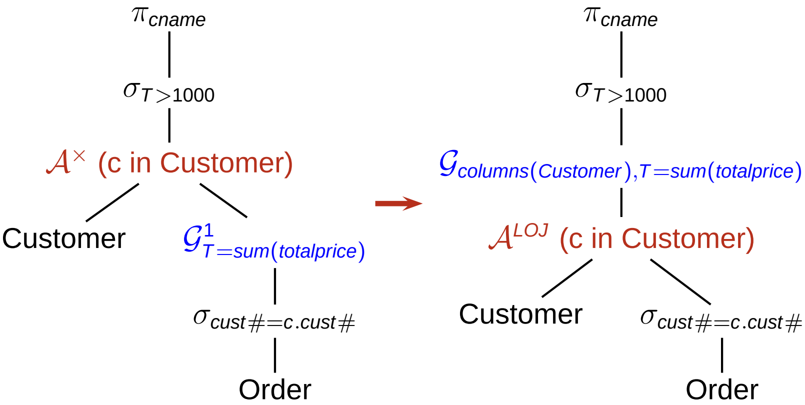
- R = relational expression
- E(r) = parameterized relational expression, $r \in R$
- $\otimes$ = relational operator that combines R and $E(r)$
  - $\otimes \in \{\times, \bowtie, \ltimes, \rtimes, \triangleright, ⟕, ⟖, ⟗\}$
- $\uplus$ = union all

Eliminating PRE with apply operator

$$\begin{aligned} Customer' &= \biguplus_{c \in Customer} (\{c\} \times E(c)) \\ &= Customer\ \mathcal{A}^{\times}\ E \end{aligned}$$



Push down apply operator



Rewriting rule: $R\ \mathcal{A}^{\times}\ (\mathcal{G}^{1}_{F}\ E) = \mathcal{G}_{columns(R),F'}\ (R\ \mathcal{A}^{LOJ}\ E)$

Example 1

Customer

cust#	cname
1	Alice
2	Bob
3	Carol

Order

order#	cust#	totalprice
100	2	3000
201	2	5000
460	3	4000
500	3	1000

Customer $\mathcal{A}^{\times}\ \mathcal{G}^{\mathbf{1}}_{\mathbf{T=sum(totalprice)}}\ \sigma_{\mathbf{cust\#=c.cust\#}}$**(Order)**

cust#	cname	T
1	Alice	null
2	Bob	8000
3	Carol	5000

R = Customer $\mathcal{A}^{\mathbf{LOJ}}\ \sigma_{\mathbf{cust\#=c.cust\#}}$**(Order)**

cust#	cname	order#	totalprice
1	Alice	null	null
2	Bob	100	3000
2	Bob	201	5000
3	Carol	460	4000
3	Carol	500	1000

$\pi_{\mathbf{cust\#,cname,T}}(\mathcal{G}_{\mathbf{columns(Customer),T=sum(totalprice)}}(\mathbf{R}))$

cust#	cname	T
1	Alice	null
2	Bob	8000
3	Carol	5000

Rewriting rule to push down apply

$$R \ \mathcal{A}^{\times} \ (\mathcal{G}^{1}_{F} \ E) \ = \ \mathcal{G}_{columns(R),F'} \ (R \ \mathcal{A}^{LOJ} \ E)$$

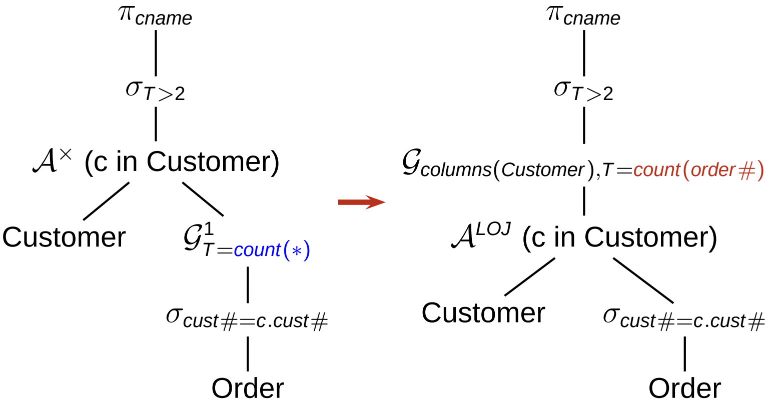
iff

(1) R must contain a key, and
(2) F' contains aggregates in F expressed over a single column

Example: If F is COUNT(*), then F' is COUNT(A) for some non-nullable column A from E

Example 2

select cname **from** Customer c **where** 2 $<$
(**select count**(*) **from** Order o **where** o.cust# = c.cust#)



Example 2 (cont.)

$$\textbf{Customer}\ \mathcal{A}^{\times}\ \mathcal{G}^{\mathbf{1}}_{\mathbf{T}=\mathbf{count}(*)}\ \sigma_{\mathbf{cust}\#=\mathbf{c.cust}\#}(\textbf{Order})$$

Customer

cust#	cname
1	Alice
2	Bob

Order

order#	cust#	date	totalprice
100	2	2012-05-03	3000
201	2	null	5000

cust#	cname	T
1	Alice	0
2	Bob	2

$$\textbf{R} = \textbf{Customer}\ \mathcal{A}^{\mathbf{LOJ}}\ \sigma_{\mathbf{cust}\#=\mathbf{c.cust}\#}(\textbf{Order})$$

cust#	cname	order#	date	totalprice
1	Alice	null	null	null
2	Bob	100	2012-05-03	3000
2	Bob	201	null	5000

$$\pi_{\mathbf{cust}\#,\mathbf{cname},\mathbf{T}}(\mathcal{G}_{\mathbf{columns}(\mathbf{Customer}),\mathbf{T}=\mathbf{count}(\mathbf{order}\#)}(\mathbf{R}))$$

cust#	cname	T
1	Alice	0
2	Bob	2

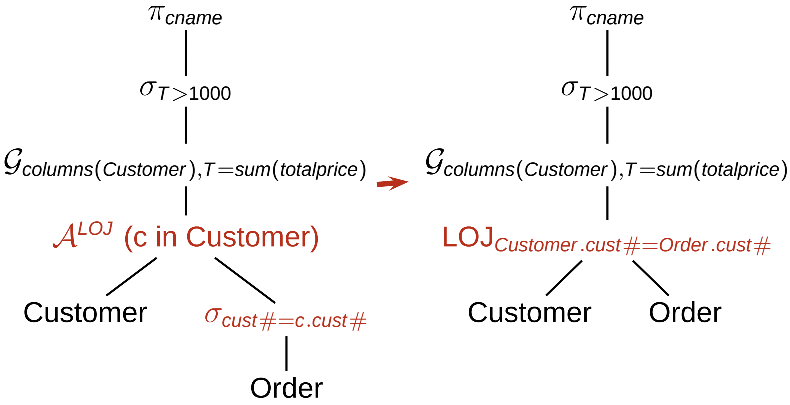
$$\pi_{\mathbf{cust}\#,\mathbf{cname},\mathbf{T}}(\mathcal{G}_{\mathbf{columns}(\mathbf{Customer}),\mathbf{T}=\mathbf{count}(*)}(\mathbf{R}))$$

cust#	cname	T
1	Alice	1
2	Bob	2

$$\pi_{\mathbf{cust}\#,\mathbf{cname},\mathbf{T}}(\mathcal{G}_{\mathbf{columns}(\mathbf{Customer}),\mathbf{T}=\mathbf{count}(\mathbf{date})}(\mathbf{R}))$$

cust#	cname	T
1	Alice	0
2	Bob	1

Eliminate apply operator



Rewriting rule: $R\ \mathcal{A}^{\otimes}\ (\sigma_p E)\ =\ R \otimes_p E$

E must not have any parameter that refers to R

Simplify LOJ to inner join

$$\pi_{cname}\left(\sigma_{T>1000}\left(\mathcal{G}_{columns(Customer),T=sum(totalprice)}\left(\text{Customer}\ \mathrm{LOJ}_{Customer.cust\#=Order.cust\#}\ \text{Order}\right)\right)\right)$$

$$\rightarrow$$

$$\pi_{cname}\left(\sigma_{T>1000}\left(\mathcal{G}_{columns(Customer),T=sum(totalprice)}\left(\text{Customer}\bowtie_{Customer.cust\#=Order.cust\#}\text{Order}\right)\right)\right)$$

Therefore ...

```
select  cname
from    Customer c
where   1000 <
        (select sum(o.totalprice)
         from   Order o
         where  o.cust# = c.cust#)
```

```
select  cname
from    (select     cname, sum(totalprice) as T
         from       Customer c join Order o
                    on c.cust# = o.cust#
         group by   cust#, cname)
where   1000 < T
```

Therefore ...

```
select  cname
from    Customer c
where   1000 <
        (select sum(o.totalprice)
         from   Order o
         where  o.cust# = c.cust#)
```

```
select    cname
from      Customer c join Order o
          on c.cust# = o.cust#
group by  c.cust#, c.cname
having    1000 < sum(o.totalprice)
```

Rewriting rules for apply operator

1. $R\ \mathcal{A}^{\otimes}\ E\ =\ R\ \otimes_{true}\ E$
2. $R\ \mathcal{A}^{\otimes}\ (\sigma_p E)\ =\ R\ \otimes_p\ E$
3. $R\ \mathcal{A}^{\times}(\sigma_p E)\ =\ \sigma_p(R\ \mathcal{A}^{\times} E)$
4. $R\ \mathcal{A}^{\times}(\pi_v E)\ =\ \pi_{v\ \cup\ columns(R)}(R\ \mathcal{A}^{\times} E)$
5. $R\ \mathcal{A}^{\times}\ (E_1\ \cup\ E_2)\ =\ (R\ \mathcal{A}^{\times}\ E_1)\ \cup\ (R\ \mathcal{A}^{\times}\ E_2)$
6. $R\ \mathcal{A}^{\times}\ (E_1\ -\ E_2)\ =\ (R\ \mathcal{A}^{\times}\ E_1)\ -\ (R\ \mathcal{A}^{\times}\ E_2)$

- Rules 1 & 2 require that E must not have any parameter that refers to R

Rewriting rules for apply operator (cont.)

7. $R\ \mathcal{A}^{\times}\ (E_1 \times E_2) = (R\ \mathcal{A}^{\times}\ E_1) \bowtie_{R.key} (R\ \mathcal{A}^{\times}\ E_2)$
8. $R\ \mathcal{A}^{\times}\ (\mathcal{G}_{A,F}\ E) = \mathcal{G}_{A\ \cup\ columns(R),F}\ (R\ \mathcal{A}^{\times}\ E)$
9. $R\ \mathcal{A}^{\times}\ (\mathcal{G}^{1}_{F}\ E) = \mathcal{G}_{columns(R),F'}\ (R\ \mathcal{A}^{LOJ}\ E)$

- Rules 7 to 9 require that R contains a key (denoted by $R.key$)
- In Rule 7, $\bowtie_{R.key}$ denote an equality join on $R.key$

References

Required Readings

- Section 4.6 (Query Tuning) of Shasha & Bonnet's book
- C.A. Galindo-Legaria, M.M. Joshi, *Orthogonal optimization of subqueries and aggregation*, SIGMOD 2001