Rewrite Rules for Relational Algebra with Applications to Query Optimization

Dirk Van Gucht[1]

[1] Indiana University

Relational Algebra Expressions

- The set of RA expressions can be recursively defined
- We will use the following notations:
  - E denotes an RA expression and $\mathbf{A}_E$ denotes its schema (i.e., set of attributes)
  - F denotes an RA expression and $\mathbf{A}_F$ denotes its schema (i.e., set of attributes)
  - C is a condition and $\mathbf{A}_C$ denotes the set of attributes that occur in C
  - L denotes an attribute list and $\mathbf{A}_L$ denotes the set of attributes in L

Relational Algebra (Recursive definition)

R	with R a relation
$(A : \mathbf{a})$	with A an attribute and $\mathbf{a}$ a constant
$E \cup F$	with $\mathbf{A}_E = \mathbf{A}_F$
$E \cap F$	with $\mathbf{A}_E = \mathbf{A}_F$
$E - F$	with $\mathbf{A}_E = \mathbf{A}_F$
$\sigma_C(E)$	with $\mathbf{A}_C \subseteq \mathbf{A}_E$
$\pi_L(E)$	with $\mathbf{A}_L \subseteq \mathbf{A}_E$
$E \times F$	with $\mathbf{A}_E \cap \mathbf{A}_F = \emptyset$
$E \bowtie_C F$	with $\mathbf{A}_E \cap \mathbf{A}_F = \emptyset$ and $\mathbf{A}_C \subseteq (\mathbf{A}_E \cup \mathbf{A}_F)$
$E \bowtie F$	
$E \ltimes F$	
$E \overline{\ltimes} F$	

Conditions (Recursive definition)

The set of conditions can be recursively defined as follows:

$A \, \theta \, \mathbf{a}$	with A an attribute, **a** a constant, and θ one of $=, \neq, <, \leq, >, \geq$
$A \, \theta \, B$	with A and B attributes and θ one of $=, \neq, <, \leq, >, \geq$
$C_1 \wedge C_2$	with C_1 and C_2 conditions
$C_1 \vee C_2$	with C_1 and C_2 conditions
$\neg \, C$	with C a condition
(C)	with C a condition

Query optimization overview

- Recall that SQL queries can be translated into equivalent RA expressions
- The benefit of this translation is that the declaratively specified SQL queries are transformed into procedurally specified queries (expressions)
- Nonetheless, these RA expressions can be inefficient to evaluate
- Rewriting these RA expressions can significantly improve this efficiency

Query optimization and SQL

- Recall that SQL can be used as a language to express RA expressions in close correspondence with RA's syntax
- Consequently, the principles for optimizing RA expressions can be applied to optimize SQL queries
- The translation algorithm from SQL to RA can be extended to incorporate optimization techniques developed for RA
- This is often a technique to improve the efficiency of SQL queries

How rewrite rules applied?

- Rewrite rules are expressed as set equalities between RA expressions. So they take the form

$$E = F$$

- Due to the property of and equality, a rewrite rule can be applied in two directions:
  1. From left to right: rewrite (replace) E by F
  2. From right to left: rewrite (replace) F by E
- For example,

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap B) \quad \text{(Distribution of } \cap \text{ over } \cup\text{)}$$

- In applying this rule, it is sometimes convenient to replace $A \cap (B \cup C)$ by $(A \cap B) \cup (A \cap C)$, while, at other times, it is useful to do this in the other direction.

How is the correctness of a rewrite rule established?

- This is often done by using the proof techniques of Predicate Logic
- For example, prove that $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.
- We need to prove that
  1. $A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$, and
  2. $(A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C)$
- To prove (1), take an element $x \in A \cap (B \cup C)$. This means that x is in A and x is in $B \cup C$
  Thus, x is in A and (a) x is in B or (b) x is in C.
  If case (a) applies, then x is in $A \cap B$ and thus x is also $(A \cap B) \cup (A \cap C)$.
  If case (b) applies, then x is in $A \cap C$ and thus x is also in $(A \cap B) \cup (A \cap C)$.
- The proof of (2) can be done in a similar way

Rewrite rules for the set operations $\cup$, $\cap$, and $-$

In this table, E, F, and G denote RA expressions with the same schemas

$E-(E-F)$	$=$	$E\cap F$	Double complementation of F relative to E
$E-(E-F)$	$=$	F	When $E\supseteq F$
$E-(F\cap G)$	$=$	$(E-F)\cup(E-G)$	Relativized De Morgan for $\cap$
$E-(F\cup G)$	$=$	$(E-F)\cap(E-G)$	Relativized De Morgan for $\cup$
$E\cap(F\cup G)$	$=$	$(E\cap F)\cup(E\cap G)$	Distribution of $\cap$ over $\cup$
$E\cup(F\cap G)$	$=$	$(E\cup F)\cap(E\cup G)$	Distribution of $\cup$ over $\cap$
$E\cap E$	$=$	E	Idempotence of $\cap$
$E\cup E$	$=$	E	Idempotence of $\cup$

Rewrite rules for the operations $\cup$, $\cap$, and $-$

In this table, E, F, and G denote RA expressions with the same schemas

$E \cap F$	$=$	$F \cap E$	Commutativity of $\cap$
$E \cup F$	$=$	$F \cup E$	Commutativity of $\cup$
$E \cap (F \cap G)$	$=$	$(E \cap F) \cap G$	Associativity of $\cap$
$E \cup (F \cup G)$	$=$	$(E \cup F) \cup G$	Associativity of $\cup$
$E \cap (E \cup F)$	$=$	E	Absorption for $\cap$
$E \cup (E \cap F)$	$=$	E	Absorption for $\cup$
$E \cap (F - E)$	$=$	$\emptyset$	Relativized contradiction for $\cap$
$E \cup (F - E)$	$=$	$E \cup F$	Relativized tautology for $\cup$
$E \cup \emptyset$	$=$	E	Identity for $\cup$
$E \cap \emptyset$	$=$	$\emptyset$	Domination for $\cap$

Rewrite rules for conditions

- Boolean conditions occur in both SQL and RA
  - In SQL, in the WHERE clause
  - In RA, in the selection operator σ
- Each of the rewrite rules for $\cup$, $\cap$, and $-$ has a corresponding logical equivalence between conditions involving OR, AND, and NOT in SQL, and involving $\vee$, $\wedge$, and $\neg$ in RA.
- For example,

$$E_1 \cap (E_2 \cup E_3) = (E_1 \cap E_2) \cup (E_1 \cap E_3)$$

$$\text{In SQL} : C_1 \text{AND} \, (C_1 \text{ OR } C_3) \leftrightarrow (C_1 \text{ AND } C_2) \text{ OR } (C_1 \text{ AND } C_3)$$

$$\text{In RA} : C_1 \wedge (C_2 \vee C_3) \leftrightarrow (C_1 \wedge C_2) \vee (C_1 \wedge C_3)$$

Rewrite rules for the selection operator σ

$\sigma_{C_1 \wedge C_2}(E)$	$=$	$\sigma_{C_1}(\sigma_{C_2}(E))$	Cascading selections
	$=$	$\sigma_{C_2}(\sigma_{C_1}(E))$	Commutativity of selections
$\sigma_{C_1 \wedge C_2}(E)$	$=$	$\sigma_{C_1}(E) \cap \sigma_{C_2}(E)$	Boolean decomposition of $\wedge$
$\sigma_{C_1 \vee C_2}(E)$	$=$	$\sigma_{C_1}(E) \cup \sigma_{C_2}(E)$	Boolean decomposition of $\vee$
$\sigma_{\neg C}(E)$	$=$	$E - \sigma_C(E)$	Boolean decomposition of $\neg$

Rewrite rules for the selection operator σ (Distribution rules)

$$
\boxed{
\begin{array}{rcl}
\sigma_C(E \cup F) & = & \sigma_C(E) \cup \sigma_C(F) \\
\\
\sigma_C(E \cap F) & = & \sigma_C(E) \cap \sigma_C(F) \\
 & = & \sigma_C(E) \cap F \\
 & = & E \cap \sigma_C(F) \\
\\
\sigma_C(E - F) & = & \sigma_C(E) - \sigma_C(F) \\
 & = & \sigma_C(E) - F
\end{array}
}
$$

Rewrite rules for interactions between selection σ and join $\bowtie$ operations

The rules of pushing down selections over joins are frequently used since they can substantially improve the efficiency of evaluating expressions!

$$
\begin{array}{lcll}
\sigma_C(E \times F) & = & E \bowtie_C F & \text{Definition of } \bowtie_C \\
 & & & \\
\sigma_C(E \bowtie F) & = & \sigma_C(E) \bowtie F & \text{when } \mathbf{A}_C \subseteq \mathbf{A}_E \\
 & = & E \bowtie \sigma_C(F) & \text{when } \mathbf{A}_C \subseteq \mathbf{A}_F \\
 & = & \sigma_C(E) \bowtie \sigma_C(F) & \text{when } \mathbf{A}_C \subseteq \mathbf{A}_E \cap \mathbf{A}_F \\
 & & & \\
\sigma_{C_1}(E \bowtie_{C_2} F) & = & \sigma_{C_1}(E) \bowtie_{C_2} F & \text{when } \mathbf{A}_{C_1} \subseteq \mathbf{A}_E \\
 & = & E \bowtie_{C_2} \sigma_{C_1}(F) & \text{when } \mathbf{A}_{C_1} \subseteq \mathbf{A}_F \\
 & = & E \bowtie_{C_1 \wedge C_2} F & \\
 & & & \\
\sigma_C(E \ltimes F) & = & \sigma_C(E) \ltimes F & \\
\sigma_C(E \mathbin{\overline{\ltimes}} F) & = & \sigma_C(E) \mathbin{\overline{\ltimes}} F &
\end{array}
$$

Rewrite rules for the interactions of projections π and the set operations $\cup$, $\cap$, and $-$

Be careful with pushing down (i.e., distributing) projections π over the set operations $\cup$, $\cap$ and $-$.

Recall the lecture on the translation algorithm from SQL to RA.

$$\boxed{\begin{array}{rcl} \pi_L(E \cup F) & = & \pi_L(E) \cup \pi_L(F) \\ \pi_L(E \cap F) & \subseteq & \pi_L(E) \cap \pi_L(F) \\ \pi_L(E - F) & \supseteq & \pi_L(E) - \pi_L(F) \end{array}}$$

Rewrite rules for the projection operator

In general, it is not the case that $\pi_L(E \cap F) = \pi_L(E) \cap \pi_L(F)$.

The following is a counter example:

R

A	B
a	b

S

A	B
a	c

In this case,

$$\pi_A(R \cap S) = \pi_A(\emptyset) = \emptyset$$

but

$$\pi_A(R) \cap \pi_A(S) = \{a\} \cap \{a\} = \{a\}$$

Rewrite rules for the projection operator π

In general, it is not the case that $\pi_L(E - F) = \pi_L(E) - \pi_L(F)$.

The following is a counter example:

R

A	B
a	b

S

A	B
a	c

$$\pi_A(R - S) = \pi_A(\{(a, b)\}) = \{a\}$$

but

$$\pi_A(R) - \pi_A(S) = \{a\} - \{a\} = \emptyset$$

Rewrite rules for the interactions of π and σ

Projection and commute

$$\boxed{\pi_L(\sigma_C(E)) = \sigma_C(\pi_L(E)) \qquad \text{when } \mathbf{A}_L \subseteq \mathbf{A}_C}$$

It is often not clear in which direction to apply this rule since both $\sigma_C(E)$ and $\pi_L(E)$ are space reducing

And it is not always clear which of these space reductions is best. This depends on the data.

Elimination rules for π

$\pi_L(E) = E$	when the schema of E corresponds precisely with L
$\pi_{B,A}(E) \neq E$	if the schema of E is (A, B) projection acts as a permutation
$\pi_{L_1}(\pi_{L_2}(E)) = \pi_{L_1}(E)$	

Rewrite rules for the interactions of projections π and joins $\bowtie$

- Observe that we can expect difficulties when we look for rules involving π and joins ($\bowtie_C$, $\bowtie$, $\ltimes$, and $\overline{\ltimes}$)
- Recall that $E \cap F = E \bowtie F$
- We know that, in general, $\pi_L(E \cap F)$ is not the same as $\pi_L(E) \cap \pi_L(F)$
- Therefore, we conclude that, in general, π does not distributes over joins

Rewrite rules for the interactions of projections π and joins $\bowtie$

We have the following important rule:

$$\boxed{\begin{array}{l} \pi_L(E \bowtie_C F) = \pi_L(\pi_{\mathbf{A}_{L_E}}(E) \bowtie_C \pi_{\mathbf{A}_{L_F}}(F)) \\ \\ \text{with } \mathbf{A}_{L_E} = \mathbf{A}_E \cap (\mathbf{A}_L \cup \mathbf{A}_C) \text{ and} \\ \text{with } \mathbf{A}_{L_F} = \mathbf{A}_F \cap (\mathbf{A}_L \cup \mathbf{A}_C) \end{array}}$$

This rule permits us to project-out (eliminate) each attribute from E (or from F) that does not appear in both

1. the projection list L and
2. the join condition C

This rule is called the attribute elimination rule or the rule of pushing projections down over joins.

Rewrite rules for π and joins $\bowtie$ (Example)

Consider

$$\pi_{a,i}(E \bowtie_{a=g} F)$$

and assume that $\mathbf{A}_E = \{a, b\}$ and $\mathbf{A}_F = \{g, h, i\}$

By the attribute elimination rule, we have that

$$\begin{aligned}
\pi_{a,i}(E \bowtie_{a=g} F) &= \pi_{a,i}(\pi_{\mathbf{A}_{L_E}}(E) \bowtie_{a=g} \pi_{\mathbf{A}_{L_F}}(F)) \\
&= \pi_{a,i}(\pi_{a}(E) \bowtie_{a=g} \pi_{g,i}(F))
\end{aligned}$$

since

$$\begin{array}{lllllll}
\mathbf{A}_{L_E} & = & \mathbf{A}_E \cap (\mathbf{A}_L \cup \mathbf{A}_C) & = & \{a, b\} \cap (\{a, i\} \cup \{a, g\}) & = & \{a\} \\
\mathbf{A}_{L_F} & = & \mathbf{A}_F \cap (\mathbf{A}_L \cup \mathbf{A}_C) & = & \{g, h, i\} \cap (\{a, i\} \cup \{a, g\}) & = & \{g, i\}
\end{array}$$

So attributes b and h were eliminated since they not appear in both the projection list and the join condition

Rewrite rule for joins $\bowtie_C$ and natural join $\bowtie$

Assume that E and F have overlapping attributes $B_1, \cdots B_k$.

Then

$$\boxed{\pi_L(E \bowtie_{E.B_1=F.B_1\wedge\cdots\wedge E.B_k=F.B_k} F) \quad = \quad \pi_L(E \bowtie F)}$$

Notice that we permit both $E.B_i$ and $F.B_i$ to simultaneously occur in L.

Of course, then the columns $E.B_i$ and $F.B_i$ in $\pi_L(R \bowtie S)$ are identical and therefore, in essence, one of them is redundant

Regular Semi-joins and Anti-semijoins

- Since regular semi-joins are combination of joins and projection, all the laws relative these operations apply.

$$\boxed{E \ltimes F \quad = \quad \pi_{\mathbf{A}_E}(E \bowtie F)}$$

- We also have the following important rewrite rule

$$\boxed{E \ltimes F \quad = \quad E \bowtie \pi_{\mathbf{A}_E \cap \mathbf{A}_F}(F)}$$

- For anti-joins, we have

$$\boxed{E \,\overline{\ltimes}\, F = E - (E \ltimes F)}$$

Thus all the laws relative to the interactions of set-difference and semi-joins apply

Example 1

- "Find the name of each student who is enrolled in course 2003."

```
SELECT  s.sname
FROM    Student s, Enroll e
WHERE   s.sid = e.sid AND e.cno = 2003
```

- This query is equivalent with the RA expression

$$\pi_{sname}(\sigma_{S.sid=E.sid \wedge cno=2003}(S \times E))$$

where S denotes Student and E denotes Enroll

Example 1 (Optimization)

- “Find the name of each student who is enrolled in course 2003."
- In RA,

$$\pi_{sname}(\sigma_{S.sid=E.sid \wedge cno=2003}(S \times E))$$

- Optimization:

$$\begin{aligned}
&\pi_{sname}(\sigma_{S.sid=E.sid \wedge cno=2003}(S \times E)) &=\\
&\pi_{sname}(\sigma_{S.sid=E.sid}(\sigma_{cno=2003}(S \times E))) &=\\
&\pi_{sname}(\sigma_{S.sid=E.sid}(S \times \sigma_{cno=2003}(E))) &=\\
&\pi_{sname}(S \bowtie_{S.sid=E.sid} \sigma_{cno=2003}(E)) &=\\
&\pi_{sname}(\pi_{sname,S.sid}(S) \bowtie_{S.sid=E.sid} \pi_{E.sid}(\sigma_{cno=2003}(E))) &=\\
&\pi_{sname}(\pi_{sname,sid}(S) \bowtie \pi_{sid}(\sigma_{cno=2003}(E))) &=\\
&\pi_{sname}(\pi_{sname,sid}(S) \ltimes \pi_{sid}(\sigma_{cno=2003}(E)))
\end{aligned}$$

Example 1 (Revisited)

- “Find the name of each student who is enrolled in course 2003."

```
SELECT   DISTINCT s.sname
FROM     Student s, Enroll e
WHERE    s.sid = e.sid AND e.cno = 2003
```

- The SQL-to-RA translation algorithm could have produced the RA expression

$$\pi_{sname}(S \bowtie \sigma_{cno=2003}(E))$$

- Optimization:

$$\begin{aligned}
&\pi_{sname}(S \bowtie \sigma_{cno=2003}(E)) &=\\
&\pi_{sname}(\pi_{sname,sid}(S) \bowtie \pi_{sid}(\sigma_{cno=2003}(E))) &=\\
&\pi_{sname}(\pi_{sname,sid}(S) \ltimes \pi_{sid}(\sigma_{cno=2003}(E)))
\end{aligned}$$

Example 1 (Revisited)

- “Find the name of each student who is enrolled in course 2003."

```
SELECT   DISTINCT s.sname
FROM     Student s, Enroll e
WHERE    s.sid = e.sid AND e.cno = 2003
```

- The SQL-to-RA translation algorithm could have produced the RA expression

$$\pi_{sname}(\sigma_{cno=2003}(S \bowtie E))$$

- Optimization:

$$\begin{aligned}
&\pi_{sname}(\sigma_{cno=2003}(S \bowtie E)) &&= \\
&\pi_{sname}(S \bowtie \sigma_{cno=2003}(E)) &&= \\
&\pi_{sname}(\pi_{sname,sid}(S) \bowtie \pi_{sid}(\sigma_{cno=2003}(E))) &&= \\
&\pi_{sname}(\pi_{sname,sid}(S) \ltimes \pi_{sid}(\sigma_{cno=2003}(E)))
\end{aligned}$$

Example 2

- "Find the sid of each student who is enrolled in at least one CS course."

```
SELECT  s.sid
FROM    Student s
WHERE   EXISTS (SELECT 1
                FROM   Enroll e, Course c
                WHERE  s.sid = e.sid AND e.cno = c.cno AND
                       dept = 'CS' )
```

- This query is equivalent with the RA expression

$$\pi_{S.sid}(\sigma_{S.sid=E.sid \wedge E.cno=C.cno \wedge dept=\text{'CS'}}(S \times E \times C))$$

where S denotes Student, E denotes Enroll, and C denotes Course

Example 2

- "Find the sid of each student who is enrolled in at least one CS course."
- Optimization:

$$
\begin{array}{ll}
\pi_{S.sid}(\sigma_{S.sid=E.sid \wedge E.cno=C.cno \wedge dept=\text{'CS'}}(S \times E \times C)) & = \\
\pi_{S.sid}(\sigma_{S.sid=E.sid}(\sigma_{E.cno=C.cno}(\sigma_{dept=\text{'CS'}}(S \times E \times C)))) & = \\
\pi_{S.sid}(S \bowtie_{S.sid=E.sid} (E \bowtie_{E.cno=C.cno} \sigma_{dept=\text{'CS'}}(C))) & = \\
\pi_{sid}(S \bowtie (E \bowtie \sigma_{dept=\text{'CS'}}(C))) & = \\
\pi_{sid}(\pi_{sid}(S) \bowtie \pi_{sid}(\pi_{sid,cno}(E) \bowtie \pi_{cno}(\sigma_{dept=\text{'CS'}}(C)))) & = \\
\pi_{sid}(S) \bowtie \pi_{sid}(\pi_{sid,cno}(E) \bowtie (\pi_{cno}(\sigma_{dept=\text{'CS'}}(C)))) & = \\
\pi_{sid}(S) \cap \pi_{sid}(\pi_{sid,cno}(E) \ltimes (\pi_{cno}(\sigma_{dept=\text{'CS'}}(C)))) & \color{red}{=} \\
\color{red}{\pi_{sid}(\pi_{sid,cno}(E) \ltimes \pi_{cno}(\sigma_{dept=\text{'CS'}}(C)))} &
\end{array}
$$

- The last equality follows since

$$\pi_{sid}(\pi_{sid,cno}(E) \ltimes (\pi_{cno}(\sigma_{dept=\text{'CS'}}(C)))) \subseteq \pi_{sid}(S)$$

This is because *sid* is a foreign key in Enroll referencing the primary key *sid* in Student.

Example 2 (Revisited)

- "Find the sid of each student who is enrolled in at least one CS course."

```
SELECT    s.sid
FROM      Student s
WHERE     EXISTS (SELECT 1
                  FROM    Enroll e, Course c
                  WHERE   s.sid = e.sid AND e.cno = c.cno AND
                          dept = 'CS' )
```

- The SQL-to-RA algorithm could have produced the RA expression:

$$\pi_{sid}(S \bowtie (E \bowtie \sigma_{dept=\text{`CS'}}(C)))$$

- Optimization:

$$\begin{array}{ll}
\pi_{sid}(S \bowtie (E \bowtie \sigma_{dept=\text{`CS'}}(C))) & = \\
\pi_{sid}(\pi_{sid}(S) \bowtie \pi_{sid}(\pi_{sid,cno}(E) \bowtie \pi_{cno}(\sigma_{dept=\text{`CS'}}(C)))) & = \\
\pi_{sid}(S) \bowtie \pi_{sid}(\pi_{sid,cno}(E) \bowtie (\pi_{cno}(\sigma_{dept=\text{`CS'}}(C)))) & = \\
\pi_{sid}(S) \cap \pi_{sid}(\pi_{sid,cno}(E) \ltimes (\pi_{cno}(\sigma_{dept=\text{`CS'}}(C)))) & = \\
\pi_{sid}(\pi_{sid,cno}(E) \ltimes \pi_{cno}(\sigma_{dept=\text{`CS'}}(C))) &
\end{array}$$

Example 3

- "Find the sid of each student who takes a course also taken by student with sid = s100."

```
SELECT   DISTINCT e1.sid
FROM     Enroll e1, Enroll e2
WHERE    e1.cno = e2.cno AND e2.sid = 's100'
```

- The SQL-to-RA algorithm could have produced the RA expression:

$$\pi_{E_1.sid}(E_1 \bowtie_{E_1.cno=E_2.cno} (\sigma_{sid=\text{`s100'}}(E_2)))$$

- Optimization:

$$\begin{array}{lll}
(1) & \pi_{E_1.sid}(E_1 \bowtie_{E_1.cno=E_2.cno} (\sigma_{sid=\text{`s100'}}(E_2))) & = \\
 & \pi_{E_1.sid}(\pi_{E_1.sid,E_1.cno}(E_1) \bowtie_{E_1.cno=E_2.cno} \pi_{E_2.cno}(\sigma_{E_2.sid=\text{`s100'}}(E_2))) & = \\
 & \pi_{sid}(\pi_{sid,cno}(E_1) \bowtie \pi_{cno}(\sigma_{sid=\text{`s100'}}(E_2))) & = \\
(2) & \pi_{sid}(\pi_{sid,cno}(E_1) \ltimes \pi_{cno}(\sigma_{sid=\text{`s100'}}(E_2))) & =
\end{array}$$

- Expression (1) is $O(|Enroll|^2)$ but expression (2) is just $O(|Enroll|)$
- Optimization results in order of magnitude improvement

Constraints and Optimization

- The presence of constraints introduces other optimization opportunities
- We consider primary key and foreign key constraints
- The optimizations apply to interactions of projections and the intersection, join and set difference operations.

Primary keys and distribution of projection over intersection and set difference

- Consider a primary key $\mathbf{K}$ in a relation R
- Then $\mathbf{K}$ uniquely determines the values of the tuples at each subset $\mathbf{A}$ of the set of attributes in R
- So we have $R(\mathbf{K}, \mathbf{A}, \cdots)$ with $\mathbf{K}$ the primary key
- We write

$$\mathbf{K} \to \mathbf{A}$$

- We have the following rewrite rules

$$\begin{aligned}
\pi_{\mathbf{K}}(\pi_{\mathbf{K},\mathbf{A}}(E_1) \cap \pi_{\mathbf{K},\mathbf{A}}(E_2)) &= \pi_{\mathbf{K}}(E_1) \cap \pi_{\mathbf{K}}(E_2) \\
\pi_{\mathbf{K}}(\pi_{\mathbf{K},\mathbf{A}}(E_1) - \pi_{\mathbf{K},\mathbf{A}}(E_2)) &= \pi_{\mathbf{K}}(E_1) - \pi_{\mathbf{K}}(E_2)
\end{aligned}$$

Primary keys and distribution of projection over joins, intersection, and set difference (Generalization)

- Consider the functional constraint

$$\mathbf{K} \rightarrow \mathbf{A}$$

- Then we have the following rewrite rules that eliminate the attributes in **A**

$$\begin{aligned}
\pi_{\mathbf{L}}(\pi_{\mathbf{K},\mathbf{A},\mathbf{B}_1}(E_1) \bowtie \pi_{\mathbf{K},\mathbf{A},\mathbf{B}_2}(E_2)) &= \pi_{\mathbf{L}}(\pi_{\mathbf{K},\mathbf{B}_1}(E_1) \bowtie \pi_{\mathbf{K},\mathbf{B}_2}(E_2)) \\
\pi_{\mathbf{L}}(\pi_{\mathbf{K},\mathbf{A},\mathbf{B}}(E_1) \cap \pi_{\mathbf{K},\mathbf{A},\mathbf{B}}(E_2)) &= \pi_{\mathbf{L}}(\pi_{\mathbf{K},\mathbf{B}}(E_1) \cap \pi_{\mathbf{K},\mathbf{B}}(E_2)) \\
\pi_{\mathbf{L}}(\pi_{\mathbf{K},\mathbf{A},\mathbf{B}}(E_1) - \pi_{\mathbf{K},\mathbf{A},\mathbf{B}}(E_2)) &= \pi_{\mathbf{L}}(\pi_{\mathbf{K},\mathbf{B}}(E_1) - \pi_{\mathbf{K},\mathbf{B}}(E_2))
\end{aligned}$$

Foreign keys and natural joins

- $R(\mathbf{F}, \mathbf{B})$ and $S(\mathbf{K}, \mathbf{A})$
- Let $\mathbf{F}$ be a foreign key in relation R referencing the primary key $\mathbf{K}$ of relation S
- Let $L \subseteq \mathbf{F} \cup \mathbf{B}$ and $\mathbf{B} \cap \mathbf{A} = \emptyset$
- We then have the following rewrite rule

$$\pi_L(R \bowtie S) = \pi_L(S \bowtie R) = \pi_L(R)$$

- So the relation S has been eliminated from this expression
- Indeed, we have

$$\begin{array}{lcll}
\pi_L(R \bowtie S) & = & \pi_L(\pi_{L \cup \mathbf{F}}(R) \bowtie \pi_{\mathbf{F}}(S)) & \text{pushing } \pi \text{ over } \bowtie \\
& = & \pi_L(\pi_{L \cup \mathbf{F}}(R)) & \mathbf{F} \text{ is a foreign key thus} \\
& & & \text{each tuple in } \pi_{L \cup \mathbf{F}}(R) \\
& & & \text{survives the join} \\
& = & \pi_L(R) &
\end{array}$$

Foreign keys and natural joins (Example)

- *Enroll*(**sid**, *cno*, *grade*) and *Student*(**<u>sid</u>**, *sname*, *byear*)
- *sid* in Enroll is a FK referencing the primary key *sid* in Student

$$\pi_{sid,cno}(Enroll \bowtie Student) = \pi_{sid,cno}(Student \bowtie Enroll) = \pi_{sid,cno}(E)$$

$$\pi_{sid}(Enroll \bowtie Student) = \pi_{sid}(Student \bowtie Enroll) = \pi_{sid}(E)$$

$$\pi_{cno}(Enroll \bowtie Student) = \pi_{cno}(Student \bowtie Enroll) = \pi_{cno}(E)$$

- The relation Student has been eliminated in these expressions. I.e., the join $\bowtie$ does not need to be done

Example 4

- “Find the cno of each course in which no students are enrolled."

```
SELECT    cno
FROM      Course
WHERE     cno NOT IN (SELECT cno
                      FROM   Enroll )
```

- After applying the translation algorithm, we get the SQL query

```
SELECT    cno
FROM      (SELECT cno, cname, dept
           FROM   Course c
           EXCEPT
           SELECT cno, cname, dept
           FROM   Course NATURAL JOIN Enroll) q
```

Example 4 (Constraints)

- "Find the cno of each course in which no students are enrolled."

```
SELECT    cno
FROM      (SELECT cno, cname, dept
           FROM    Course
           EXCEPT
           SELECT cno, cname, dept
           FROM    Course NATURAL JOIN Enroll) q
```

- This gets translated to the RA expression

$$\begin{array}{lll} (1) & \pi_{cno}(\pi_{cno,cname,dept}(C) - \pi_{cno,cname,dept}(C \bowtie E)) & = \\ (2) & \pi_{cno}(C) - \pi_{cno}(C \bowtie E) & = \\ (3) & \pi_{cno}(C) - \pi_{cno}(E) & \end{array}$$

- The rule that takes (1) to (2) follows since *cno* is a primary key for Course
- The rule that takes (2) to (3) follows since *cno* is foreign key in Enroll referencing the primary key *cno* in Course

Example 5

“Find the sid of each student who is only enrolled CS courses."

```
SELECT    s.sid
FROM      Student s
WHERE     NOT EXISTS (SELECT 1
                      FROM   Enroll e
                      WHERE e.sid = s.sid AND
                       e.cno NOT IN (SELECT c.cno
                                     FROM   Course c
                                     WHERE  c.dept = ‘CS’))
```

Example 5

“Find the sid of each student who is only enrolled in CS courses."

Using the translation algorithm this becomes the SQL query

```
WITH      CS AS (SELECT * FROM Course WHERE dept = 'CS')
SELECT    q1.ssid
FROM       (SELECT s.sid AS ssid, s.sname
            FROM    Student s
            EXCEPT
            SELECT q2.ssid, s.sname
            FROM (SELECT s.sid AS ssid, s.sname, e.sid, e.cno
                  FROM    Student s NATURAL JOIN Enroll e
                  EXCEPT
                  (SELECT s.sid, s.sname, e.sid, e.cno
                   FROM    Student s CROSS JOIN
                           Enroll e NATURAL JOIN CS ) q2) q1
```

is correspond to the RA expression

$$\pi_{sid}(\pi_{sid,sname}(S) - \pi_{sid,sname}(\pi_{S.sid,sname,E.sid,cno}(S \bowtie E) - \pi_{S.sid,sname,E.sid,cno}(S \bowtie E \bowtie \sigma_{dept=\text{'}CS\text{'}}(C))))$$

where S, E, and C denote Student, Enroll, and Course, respectively.

Example 5

"Find the sid of each student who is only enrolled in CS courses."

$$\pi_{sid}(\pi_{sid,sname}(S) - \pi_{sid,sname}(\pi_{S.sid,sname,E.sid,cno}(S \bowtie E) - \pi_{S.sid,sname,E.sid,cno}(S \bowtie E \bowtie \sigma_{dept='CS'}(C))))$$

Because *sid* is a primary key of Student and we do not need *sname*, we can rewrite this expression to

$$\pi_{sid}(S) - \pi_{sid}(\pi_{S.sid,E.sid,cno}(S \bowtie E) - \pi_{S.sid,E.sid,cno}(S \bowtie E \bowtie \sigma_{dept='CS'}(C)))$$

Because *sid* is a FK in Enroll referencing the primary key *sid* in Student, we can rewrite this expression to

$$\pi_{sid}(S) - \pi_{sid}(\pi_{sid,cno}(E) - \pi_{sid,cno}(E \ltimes \sigma_{dept='CS'}(C)))$$

If Enroll has schema (*sid*, *cno*) then this is

$$\pi_{sid}(S) - \pi_{sid}(E - E \ltimes \pi_{cno}(\sigma_{dept='CS'}(C)))$$

Example 5

"Find the sid of each student who is only enrolled CS courses."

```
SELECT   s.sid
FROM     Student s
WHERE    NOT EXISTS (SELECT 1
                     FROM   Enroll e
                     WHERE e.sid = s.sid AND
                      e.cno NOT IN (SELECT c.cno
                                    FROM    Course c
                                    WHERE   c.dept = 'CS'))
```

is translated and optimized to

$$\pi_{sid}(S) - \pi_{sid}(E - E \ltimes \pi_{cno}(\sigma_{dept='CS'}(C)))$$

Or, more succinctly, using the anti-semijoin

$$\pi_{sid}(S) - \pi_{sid}(E \overline{\ltimes} \pi_{cno}(\sigma_{dept='CS'}(C)))$$