University of Southampton

Relational Algebra 2

COMP3211 Advanced Databases

Dr Heather Packer – hp3@ecs.soton.ac.uk

Recap

- Set-theoretic bases for Relational Algebra
- Set Operations
  - Union, Difference, Cartesian Product
- Relational Operations
  - Renaming, Projection, Selection
- Sets and Multisets

Commutativity **does not** hold for Cartesian Product named

$R \times S \neq S \times R$

Name
Union
Co-op
Costa

x

Addr
Campus
Burgess Road
Burgess Road

$\neq$

Addr
Campus
Burgess Road
Burgess Road

x

Name
Union
Co-op
Costa

Name	Addr
Union	Campus
Co-op	Burgess Road
Costa	Burgess Road

$\neq$

Addr	Name
Campus	Union
Burgess Road	Co-op
Burgess Road	Costa

Commutativity **does not** hold for Cartesian Product unnamed

$R \times S \neq S \times R$

Union	x	Campus	≠	Campus	x	Union
Co-op		Burgess Road		Burgess Road		Co-op
Costa		Burgess Road		Burgess Road		Costa

Union	Campus
Co-op	Burgess Road
Costa	Burgess Road

≠

Campus	Union
Burgess Road	Co-op
Burgess Road	Costa

Relational Database binary operations

- Binary Operators between two relations
- Used to combine information from 2 relations into a new relation
- Core to Relational Databases

Θ-Join $\bowtie_F$

- Theta Join combines two relations using a predicate F

$$R \bowtie_F S$$

- It is equivalent to the cartesian product of the two relations followed by a selection using the predicate:

$$\sigma_F(R \times S)$$

- It is called a “theta join” because in the original notation, Θ was used in place of F and was limited to:

  =, <, >, <=, >=, !=

- A theta join that only uses the operator = is called an Equijoin

Θ-Join Example

Food

Shop	Food	Price	Units
Union	Apples	0.50	2
Union	Bananas	0.80	4
Co-op	Apples	0.50	5
Co-op	Peaches	0.75	3
Costa	Bananas	0.90	1
Costa	Peaches	1.10	1

Locations

Name	Addr
Union	Campus
Co-op	Burgess Road
Costa	Burgess Road

Food $\bowtie_{\text{Food.Shop=Locations.Name}}$ Locations

This is also an Equijoin

Shop	Food	Price	Units	Name	Addr
Union	Apples	0.50	2	Union	Campus
Union	Bananas	0.80	4	Union	Campus
Co-op	Apples	0.50	5	Co-op	Burgess Road
Co-op	Peaches	0.75	3	Co-op	Burgess Road
Costa	Bananas	0.90	1	Costa	Burgess Road
Costa	Peaches	1.10	1	Costa	Burgess Road

Natural Join

- An **natural join** is a Θ-join in which no predicate is specified

$$R \bowtie S$$

- It is defined an equijoin over all the **common** attributes of the two relations
- The result contains the common attributes followed by the remaining non-common attributes in R and S
  - like an equijoin but the common attributes only appear once

Natural Join Example - Food ⋈ Locations

Renamed Name to Shop to make this work

Food

Shop	Food	Price	Units
Union	Apples	0.50	2
Union	Bananas	0.80	4
Co-op	Apples	0.50	5
Co-op	Peaches	0.75	3
Costa	Bananas	0.90	1
Costa	Peaches	1.10	1

Locations

Shop	Addr
Union	Campus
Co-op	Burgess Road
Costa	Burgess Road

Food ⋈ Locations =

Food	Price	Units	Shop	Addr
Apples	0.50	2	Union	Campus
Bananas	0.80	4	Union	Campus
Apples	0.50	5	Co-op	Burgess Road
Peaches	0.75	3	Co-op	Burgess Road
Bananas	0.90	1	Costa	Burgess Road
Peaches	1.10	1	Costa	Burgess Road

Common attributes: Shop
Non-common attributes:
Food, Price, Units, Addr

Natural Join ⋈

- Natural Join can be formalised as the Cartesian Product of R and S, followed by the selection on equality amongst the common attributes (A_1,.. A_k). Followed by a projection.

$$R \bowtie S = \pi_{<list>}(\sigma_{R.A1=S.A1 \land \ldots \land R.Ak = S.Ak}(R \times S))$$

- where <list> contains
  - All the attributes unique to R
  - All the common attributes
  - All the attributes unique to S

Natural join ⋈ Example

- Given relations:
  - REGISTERED(student, course, term)
  - TEACHES(lecturer, course, term)

$$\text{REGISTERED} \bowtie \text{TEACHES}$$

$$= \text{TAUGHT}(\text{student}, \text{course}, \text{term}, \text{lecturer})$$

Left Outer Join ⟕

The Left outer join of two relations R and S is a natural join which also includes tuples from R which do not have corresponding tuples in S; missing values are set to null

R

A	B
a	1
b	2

S

B	C
1	x
1	y
3	z

R ⟕ S

A	B	C
a	1	x
a	1	Y
b	2	null

$$R ⟕ S = R \bowtie S \cup ((R - \pi_{r1,\, r2,\ldots,rn}(R \bowtie S)) \times \{<null,\ldots,\, null>\})$$

Attributes in R

A relation on attributes in S but not in R, that contains a single tuple

A tuple of nulls with same arity as S

Outer Join

Left Outer Join	R ⟕ S	$((R - \pi_{r1,\ r2,\ldots,rn}(R \bowtie S)) \times \{\}) \cup R \bowtie S$
Right Outer Join	R ⟖ S	$((S - \pi_{s1,\ s2,\ldots,sn}(R \bowtie S)) \times \{\}) \cup R \bowtie S$
Full Outer Join	R ⟗ S	$(((R - \pi_{r1,\ r2,\ldots,rn}(R \bowtie S)) \times \{\}) \cup ((S - \pi_{s1,\ s2,\ldots,sn}(R \bowtie S)) \times \{\}) \cup R \bowtie S)$

Semijoin $\ltimes$

Semijion is like a natural join but the resulting attributes are only taken from A

$$R \ltimes S$$

R

A	B
a	1
b	2

S

B	C
1	x
1	y
3	z

$R \ltimes S$

A	B
a	1

$$R \ltimes S \equiv \pi_L (R \bowtie S)$$

where L is the list of attributes in R

Antijoin $\triangleright$

- The antijoin is like semijoin but the result only contains tuples from R that have **no** match in S

$$R \triangleright S$$

R

A	B
a	1
b	2

S

B	C
1	x
1	y
3	z

$R \triangleright S$

A	B
b	2

$$R \triangleright S \equiv R - (R \ltimes S)$$

Relational Transformations

Relational Transformations

- Relational expressions can be transformed with transformation rules
- Used during SQL query optimisation to rewrite user queries
- Database engine aims to improve CPU, memory or disk usage

Relational Transformations

- When an expression consists of a series of nested projections, only the last in a sequence of projections is required

$$\pi_L \pi_M \ldots \pi_N(R) = \pi_L(R)$$

- But not if they are extended projections that rely on prior expressions

Relational Transformations

- If a selection contains a predicate with conjunctive terms (ie ANDs)
- The terms can cascade into individual selections

$$\sigma_{p \land q \land r}(R) = \sigma_p(\sigma_q(\sigma_r(R)))$$

Relational Transformations - commutative

- Selection and theta-join are commutative operations

$$\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$$

$$R \bowtie_p S = S \bowtie_p R$$

- But for theta-join, only when using the named perspective
  - Eg using attribute names and not $1 etc

Relational Transformations - commutative

- When an expression consists of a selection followed by a projection
- The projection can be done first, if the selection predicate only involves attributes in the projection list:

$$\pi_{A1,\ldots Am}(\sigma_p(R)) = \sigma_p(\pi_{A1,\ldots Am}(R))$$

- Selection and projection are commutative

Relational Transformations - associativity

- Joins exhibits associativity:

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

Relational Transformations - distributes

- Where an expression consists of a theta-join followed by a projection
- The selection can be performed on both relations prior to the theta-join, if the predicate only involves attributes being joined

$$\sigma_p(R \bowtie_r S) = \sigma_p(R) \bowtie_r \sigma_p(S)$$

- In this case, selection distributes over theta-join

Relational Transformations - distributes

- Selection also distributes over set operations

$$\sigma_p(R \cup S) = \sigma_p(R) \cup \sigma_p(S)$$

$$\sigma_p(R \cap S) = \sigma_p(R) \cap \sigma_p(S)$$

$$\sigma_p(R - S) = \sigma_p(R) - \sigma_p(S)$$

Relational Transformations - distributes

- Projection distributes over set union

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

Relational Transformations - distributes

- Projection distributes over theta join

$$\pi_{L1 \cup L2}(R \bowtie_r S) = \pi_{L1}(R) \bowtie_r \pi_{L2}(S)$$

if projection list can be divided into attributes of the relations being joined, and join condition only uses attributes from the projection list

Relational Algebra and SQL

Relational Transformations

- A Basic SQL statement consists of the following form:

  SELECT $R_{i1}.A_1, \ldots, R_{im}.A_m$

  FROM $R_1, \ldots, R_K$

  WHERE Θ

- $R_1, \ldots, R_K$ are distinct relation names (no repetitions)
- Each $R_{ij}.A_j$ is an attribute of R_{ij} $(1 \leq ij \leq k)$
- Θ is a condition

SQL vs Relational Algebra

SQL	Relational Algebra
SELECT	Projection π
FROM	Cartesian Product
WHERE	Selection σ

SELECT $R_{i1}.A_1, \ldots, R_{im}.A_m$

FROM $R_1, \ldots, R_K$ $\quad = \pi_{Ri1.A1, \ldots, Rim.Am}(\sigma_\Theta(R_1 \times \ldots \times R_K))$

WHERE Θ

SELECT * no projection operator is used i.e. expression is $\sigma_\Theta(R_1 \times \ldots \times R_K)$

SQL vs Relational Algebra Examples

DB Schema: FACULTY(name, dpt, salary), CHAIR(dpt, name)

Query: Find the salaries of department chairs

C-SALARY(dpt,salary) =

Relational Algebra:

$$\pi_{\text{F.dpt, F.salary}}(\sigma_{\text{F.name = C.name} \land \text{F.dpt = C.dpt}}(\text{FACULTY} \times \text{CHAIR}))$$

also

$$\pi_{\text{dpt, salary}}(\text{FACULTY} \bowtie \text{CHAIR})$$

SQL vs Relational Algebra Examples

C-SALARY(dpt,salary) =

$$\pi_{F.dpt,\ F.salary}(\sigma_{F.name\ =\ C.name\ \wedge\ F.dpt\ =\ C.dpt}\ (FACULTY \times CHAIR))$$

SQL:

```
SELECT FACULTY.dpt, FACULTY.salary
FROM FACULTY, CHAIR
WHERE FACULTY.name = CHAIR.name AND FACULTY.dpt = CHAIR.dpt
```

SQL vs Relational Algebra Examples - No Selection

- Goal: Compute the Cartesian product of relations of S and T
- Relational algebra: $S \times T$
- SQL:

```
SELECT *
FROM S, T
```

- WHERE clause is not always necessary in SQL
  - E.g., when having a relational algebra query with no selection operation

SQL vs Relational Algebra Examples - Self-Joins

- Goal: Compute expressions that rely on Self-Joins
- However, relation names in the FROM list must be distinct
- This stops us from computing self-joins, ie FROM R, R
- Many interesting queries involve self-joins

Self-Join Example

- DB Schema:

  FATHER(father-name, child-name)

- Compute

  GRANDFATHER(grandfather-name, grandchild-name)

First take the Cartesian Product of Father and Father

FATHER

father-name	child-name
David	Nick
Nick	Joe
Joe	Mick

x

FATHER

father-name	child-name
David	Nick
Nick	Joe
Joe	Mick

Self-Joins

FATHER x FATHER

father-name	child-name	father-name	child-name
David	Nick	David	Nick
Nick	Joe	David	Nick
Joe	Mick	David	Nick
David	Nick	Nick	Joe
Nick	Joe	Nick	Joe
Joe	Mick	Nick	Joe
David	Nick	Joe	Mick
Nick	Joe	Joe	Mick
Joe	Mick	Joe	Mick

Second, select where the child-name is the same as the father-name

$\sigma_{\$2=\$3}$ (FATHER x FATHER)

father-name	child-name	father-name	child-name
David	Nick	Nick	Joe
Nick	Joe	Joe	Mick

Self-Joins

$\sigma_{\$2=\$3}$ (FATHER x FATHER)

father-name	child-name	father-name	child-name
David	Nick	Nick	Joe
Nick	Joe	Joe	Mick

Project the first and last attributes

$\pi_{\$1,\$4}$ ($\sigma_{\$2=\$3}$ (FATHER x FATHER))

father-name	child-name
David	Joe
Nick	Mick

Self-Joins

$\pi_{\$1,\$4} (\sigma_{\$2=\$3} (\text{FATHER} \times \text{FATHER}))$

father-name	child-name
David	Joe
Nick	Mick

Rename the attributes to grandfather-name and grandchild-name

$\rho_{\text{grandfather-name/father-name, grandchild-name/child-name}} (\pi_{\$1,\$4} (\sigma_{\$2=\$3} (\text{FATHER} \times \text{FATHER})))$

grandfather-name	grandchild-name
David	Joe
Nick	Mick

GRANDFATHER(grandfather-name, grandchild-name)

Self-Joins

- In relational algebra, we can reference columns by position number
  - Eg in our expression for GRANDFATHER:

$\rho_{\text{father-name/grandfather-name, child-name/grandchild-name}} (\pi_{\$1,\$4} (\sigma_{\$2=\$3} (\text{FATHER} \times \text{FATHER})))$

- SQL does not support referencing columns by position number
- Instead, SQL supports an aliasing mechanism

Aliases in SQL

- SQL allows us to give one or more new names to a relation
  - these are aliases of the given relation
- Rules for Aliases Creation
  - Aliases are created in the FROM list
    - FROM <relation name> AS <renamed relation name>, …
  - The new names can be referenced in the SELECT list and in the WHERE clause

Example:

– Expressing $R \times R$ in SQL:

```
SELECT *
FROM R AS S, R AS T
```

Aliases in SQL

- DB Schema: FATHER(father-name,child-name)
- Compute

GRANDFATHER(grandfather-name,grandchild-name) in SQL:

```
SELECT R.father-name AS grandfather-name, T.child-name AS grandchild-name
FROM FATHER AS R, FATHER AS T
WHERE R.child-name = T.father-name
```

- SQL allows for the renaming of attribute names in the SELECT list
- Aliases in SQL are used not only out of necessity, but also for convenience in order to create short nicknames for relations.

Relational Completeness of SQL

- SQL can express all relational algebra queries
  - (i.e., it is a relationally complete database query language)
- As we saw

```
SELECT DISTINCT …
FROM …
WHERE …
```

- can express Cartesian product, projection, and selection

Relational Completeness of SQL

- SQL has explicit constructs for union and difference:
- Union R ∪ S:

  (SELECT * FROM R) UNION (SELECT * FROM S)
- Difference R – S:

  (SELECT * FROM R) EXCEPT (SELECT * FROM S)
- UNION and EXCEPT eliminates duplicates! (Set semantics)
- UNION ALL and EXCEPT ALL does not (Multiset semantics)

Next Lecture:
Query Processing