

Lecture 14: More Relational Algebra

Selection, some definitions and hacks

- select
- joins: theta, equi, natural, outer
- union compatibility, and the set operations for relational algebra
- intermediate results and attribute renaming

The Query

```
SELECT enumber, fname, lname, dname
FROM employee, department
WHERE (dname='Administration' OR dname='Finance')
AND dnumber=dno;
```

So far we have made sense of the `SELECT ...FROM ...` part of this query.

The Query

```
SELECT enumber, fname, lname, dname
FROM employee, department
WHERE (dname='Administration' OR dname='Finance')
AND dnumber=dno;
```

So far we have made sense of the `SELECT ...FROM ...` part of this query.

Now we are ready to work on the `WHERE ...` part.

Condition Expressions

- Each tuple in a relation gives a value to each index, or attribute name.

Condition Expressions

- Each tuple in a relation gives a value to each index, or attribute name.
- So, expressions like `dname = 'Administration'` can be evaluated to *true* or *false* in the context of a tuple.

Condition Expressions

- Each tuple in a relation gives a value to each index, or attribute name.
- So, expressions like `dname = 'Administration'` can be evaluated to *true* or *false* in the context of a tuple.
- Most of the domains we use are totally ordered, so we can also evaluate eg. `dname < 'Administration'` or `enumber \geq 100`

Condition Expressions

- Each tuple in a relation gives a value to each index, or attribute name.
- So, expressions like `dname = 'Administration'` can be evaluated to *true* or *false* in the context of a tuple.
- Most of the domains we use are totally ordered, so we can also evaluate eg. `dname < 'Administration'` or `enumber \geq 100`
- These atomic expressions can be combined in the obvious way using boolean connectives AND (\wedge , $\&$), OR (\vee) and NOT (\neg , \sim)

Condition Expressions

- Each tuple in a relation gives a value to each index, or attribute name.
- So, expressions like `dname = 'Administration'` can be evaluated to *true* or *false* in the context of a tuple.
- Most of the domains we use are totally ordered, so we can also evaluate eg. `dname < 'Administration'` or `enumber \geq 100`
- These atomic expressions can be combined in the obvious way using boolean connectives AND (\wedge , $\&$), OR (\vee) and NOT (\neg , \sim)
- Any such condition expression defines a function from the relation to $\{true, false\}$.

SQL's `WHERE` is Relational Algebra's *select*

We use the condition expression to “filter” a relation, keeping only the tuples that evaluate to *true*.

SQL's WHERE is Relational Algebra's *select*

We use the condition expression to “filter” a relation, keeping only the tuples that evaluate to *true*.

Here's the notation and definition:

$$\sigma_{cond}(R) = \{t \in R \mid cond(t) = true\}$$

Facts about Select

- Conjunction (and) is composition is intersection

$$\sigma_{c \wedge c'}(R) = \sigma_c(\sigma_{c'}(R)) = \sigma_{c'}(\sigma_c(R)) = \sigma_c(R) \cap \sigma_{c'}(R)$$

Facts about Select

- Conjunction (and) is composition is intersection

$$\sigma_{c \wedge c'}(R) = \sigma_c(\sigma_{c'}(R)) = \sigma_{c'}(\sigma_c(R)) = \sigma_c(R) \cap \sigma_{c'}(R)$$

- $\sigma_{c \vee c'}(R) = \sigma_c(R) \cup \sigma_{c'}(R)$

Facts about Select

- Conjunction (and) is composition is intersection
$$\sigma_{c \wedge c'}(R) = \sigma_c(\sigma_{c'}(R)) = \sigma_{c'}(\sigma_c(R)) = \sigma_c(R) \cap \sigma_{c'}(R)$$
- $\sigma_{c \vee c'}(R) = \sigma_c(R) \cup \sigma_{c'}(R)$
- $\sigma_{\neg c}(R) = R - \sigma_c(R)$

Facts about Select

- Conjunction (and) is composition is intersection
$$\sigma_{c \wedge c'}(R) = \sigma_c(\sigma_{c'}(R)) = \sigma_{c'}(\sigma_c(R)) = \sigma_c(R) \cap \sigma_{c'}(R)$$
- $\sigma_{c \vee c'}(R) = \sigma_c(R) \cup \sigma_{c'}(R)$
- $\sigma_{\neg c}(R) = R - \sigma_c(R)$
- $|\sigma_c(R)| \leq |R|$

Facts about Select

- Conjunction (and) is composition is intersection
$$\sigma_{c \wedge c'}(R) = \sigma_c(\sigma_{c'}(R)) = \sigma_{c'}(\sigma_c(R)) = \sigma_c(R) \cap \sigma_{c'}(R)$$
- $\sigma_{c \vee c'}(R) = \sigma_c(R) \cup \sigma_{c'}(R)$
- $\sigma_{\neg c}(R) = R - \sigma_c(R)$
- $|\sigma_c(R)| \leq |R|$

Facts about Select

- Conjunction (and) is composition is intersection
$$\sigma_{c \wedge c'}(R) = \sigma_c(\sigma_{c'}(R)) = \sigma_{c'}(\sigma_c(R)) = \sigma_c(R) \cap \sigma_{c'}(R)$$
- $\sigma_{c \vee c'}(R) = \sigma_c(R) \cup \sigma_{c'}(R)$
- $\sigma_{\neg c}(R) = R - \sigma_c(R)$
- $|\sigma_c(R)| \leq |R|$

The left-hand sides might be faster to process, since they pass through the relation just once.

Facts about Select

- Conjunction (and) is composition is intersection
$$\sigma_{c \wedge c'}(R) = \sigma_c(\sigma_{c'}(R)) = \sigma_{c'}(\sigma_c(R)) = \sigma_c(R) \cap \sigma_{c'}(R)$$
- $\sigma_{c \vee c'}(R) = \sigma_c(R) \cup \sigma_{c'}(R)$
- $\sigma_{\neg c}(R) = R - \sigma_c(R)$
- $|\sigma_c(R)| \leq |R|$

The left-hand sides might be faster to process, since they pass through the relation just once.

All depends on the access paths (index, hash) into the relation vs attributes used in query. The algebra gives us options to assess.

The Query as Relational Algebra

$$\pi_{fname, lname, dname} \left(\sigma_{dname='Finance' \vee dname='Administration'} \left(\sigma_{dnum=dno} (employee \times department) \right) \right)$$

I have expressed it with two separate selects, one for the join, one for the user requirement.

Join

Joining two tables using product and selection is so common, we define a new operation for it.

$$A \bowtie_{a=b} B \equiv \sigma_{a=b}(A \times B)$$

Join

Joining two tables using product and selection is so common, we define a new operation for it.

$$A \bowtie_{a=b} B \equiv \sigma_{a=b}(A \times B)$$

Using this, our query becomes

$$\pi_{fname, lname, dname} \left(\sigma_{dname='Finance' \vee dname='Administration'} \left(\mathit{employee} \bowtie_{dnum=dno} \mathit{department} \right) \right)$$

Join

Joining two tables using product and selection is so common, we define a new operation for it.

$$A \bowtie_{a=b} B \equiv \sigma_{a=b}(A \times B)$$

Using this, our query becomes

$$\pi_{fname, lname, dname} \left(\sigma_{dname='Finance' \vee dname='Administration'} \left(\mathit{employee} \bowtie_{dnum=dno} \mathit{department} \right) \right)$$

Its also possible to code SQL in this style

```
SELECT fname, lname, dname
FROM (employee JOIN department ON dum=dno) WHERE
(dname='Finance' OR dname='Administration');
```

Theta, Equi and Natural Joins

If the condition c on the join $A \bowtie_c B$ is $c_1 \wedge c_2 \wedge \dots$ where each c_i is a comparison ($\leq, <, =, >, \geq$) between an A attribute and a B attribute, the join is called a *theta* join.

Theta, Equi and Natural Joins

If the condition c on the join $A \bowtie_c B$ is $c_1 \wedge c_2 \wedge \dots$ where each c_i is a comparison ($\leq, <, =, >, \geq$) between an A attribute and a B attribute, the join is called a *theta* join.

Theta, Equi and Natural Joins

If the condition c on the join $A \bowtie_c B$ is $c_1 \wedge c_2 \wedge \dots$ where each c_i is a comparison ($\leq, <, =, >, \geq$) between an A attribute and a B attribute, the join is called a *theta* join.

If all the comparisons in a theta join are equalities ($=$), its called an *equijoin*.

Theta, Equi and Natural Joins

If the condition c on the join $A \bowtie_c B$ is $c_1 \wedge c_2 \wedge \dots$ where each c_i is a comparison ($\leq, <, =, >, \geq$) between an A attribute and a B attribute, the join is called a *theta* join.

If all the comparisons in a theta join are equalities ($=$), its called an *equijoin*.

Theta, Equi and Natural Joins

If the condition c on the join $A \bowtie_c B$ is $c_1 \wedge c_2 \wedge \dots$ where each c_i is a comparison ($\leq, <, =, >, \geq$) between an A attribute and a B attribute, the join is called a *theta* join.

If all the comparisons in a theta join are equalities ($=$), its called an *equijoin*.

A *natural join* is a special join to use in place of a common simple kind of equijoin: with each equation having an attribute from A and an attribute from B with the *same name*.

Theta, Equi and Natural Joins

If the condition c on the join $A \bowtie_c B$ is $c_1 \wedge c_2 \wedge \dots$ where each c_i is a comparison ($\leq, <, =, >, \geq$) between an A attribute and a B attribute, the join is called a *theta* join.

If all the comparisons in a theta join are equalities ($=$), its called an *equijoin*.

A *natural join* is a special join to use in place of a common simple kind of equijoin: with each equation having an attribute from A and an attribute from B with the *same name*.

Theta, Equi and Natural Joins

If the condition c on the join $A \bowtie_c B$ is $c_1 \wedge c_2 \wedge \dots$ where each c_i is a comparison ($\leq, <, =, >, \geq$) between an A attribute and a B attribute, the join is called a *theta* join.

If all the comparisons in a theta join are equalities ($=$), its called an *equijoin*.

A *natural join* is a special join to use in place of a common simple kind of equijoin: with each equation having an attribute from A and an attribute from B with the *same name*.

We write

$$A * B$$

for this, and the resulting table only has *one* (not two) of each of those attributes named in the condition.

Outer Joins

- After the review and restructure of our example organisation, there were no employees in the Finance department.

Outer Joins

- After the review and restructure of our example organisation, there were no employees in the Finance department.
- What would our query show in that situation?

Outer Joins

- After the review and restructure of our example organisation, there were no employees in the Finance department.
- What would our query show in that situation?
- The Finance department would not appear at all!

Outer Joins

- After the review and restructure of our example organisation, there were no employees in the Finance department.
- What would our query show in that situation?
- The Finance department would not appear at all!
- If we wanted to list *all the departments, and their employees* we could use an **outer join**.

Outer Joins

- After the review and restructure of our example organisation, there were no employees in the Finance department.
- What would our query show in that situation?
- The Finance department would not appear at all!
- If we wanted to list *all the departments, and their employees* we could use an outer join.
- This allows us to specify that every tuple of one, or both input relations must appear in the result.

Outer Join Example

fname	lname	dname
Jack	White	Administration Finance
Jack	Black	Computer Science
Ronnie-James	Dio	Computer Science

Outer Join Example

fname	lname	dname
Jack	White	Administration Finance
Jack	Black	Computer Science
Ronnie-James	Dio	Computer Science

If a tuple from (one of) the specified relation(s) does not get a match in the join condition, it appears with extra NULL values in the resulting table.

Union Compatibility

- Mathematically, we can form the union of *any* two sets.

Union Compatibility

- Mathematically, we can form the union of *any* two sets.
- But the result will not always be a relation.

Union Compatibility

- Mathematically, we can form the union of *any* two sets.
- But the result will not always be a relation.
- The union of two relations of the same relation schema will also belong to that schema.

Union Compatibility

- Mathematically, we can form the union of *any* two sets.
- But the result will not always be a relation.
- The union of two relations of the same relation schema will also belong to that schema.
- Various conditions weaker than “belonging to the same schema” could allow relations to be combined to form new relations.
(exercise: find two different alternatives)

Union Compatibility

- Mathematically, we can form the union of *any* two sets.
- But the result will not always be a relation.
- The union of two relations of the same relation schema will also belong to that schema.
- Various conditions weaker than “belonging to the same schema” could allow relations to be combined to form new relations.
(exercise: find two different alternatives)
- The one defined in relational algebra is called “*union compatibility*”

Union Compatibility

- Mathematically, we can form the union of *any* two sets.
- But the result will not always be a relation.
- The union of two relations of the same relation schema will also belong to that schema.
- Various conditions weaker than “belonging to the same schema” could allow relations to be combined to form new relations.
(exercise: find two different alternatives)
- The one defined in relational algebra is called “*union compatibility*”

Union Compatibility

- Mathematically, we can form the union of *any* two sets.
- But the result will not always be a relation.
- The union of two relations of the same relation schema will also belong to that schema.
- Various conditions weaker than “belonging to the same schema” could allow relations to be combined to form new relations.
(exercise: find two different alternatives)
- The one defined in relational algebra is called “*union compatibility*”

Two relations are *union compatible* if they

- have the same number of attributes
- for each **position**, the attributes in the two relations have the same domain

Set Operations in Relational Algebra

The set operations union (\cup), intersection (\cap) and set difference ($-$) can be applied to any pair of relations that are union compatible.

Set Operations in Relational Algebra

The set operations union (\cup), intersection (\cap) and set difference ($-$) can be applied to any pair of relations that are union compatible.

The resulting relation will have the attribute names from the *first* input relation.

Set Operations in Relational Algebra

The set operations union (\cup), intersection (\cap) and set difference ($-$) can be applied to any pair of relations that are union compatible.

The resulting relation will have the attribute names from the *first* input relation.

Thus, unlike true union and intersection, these operations are *not commutative*. For example, the substitutivity principle is violated - different results from the same projection.

$$A \cup B \neq B \cup A$$

([E&N §6.2] makes the obviously incorrect claim that they are commutative, probably assuming the names can be ignored.)

Set Operations in Relational Algebra

The set operations union (\cup), intersection (\cap) and set difference ($-$) can be applied to any pair of relations that are union compatible.

The resulting relation will have the attribute names from the *first* input relation.

Thus, unlike true union and intersection, these operations are *not commutative*. For example, the substitutivity principle is violated - different results from the same projection.

$$A \cup B \neq B \cup A$$

([E&N §6.2] makes the obviously incorrect claim that they are commutative, probably assuming the names can be ignored.)

Set difference is not commutative, even in set-theory.

Intermediate Results and Renaming

- To write out the relational algebra forms of the example query, we had to break it into several lines.

Intermediate Results and Renaming

- To write out the relational algebra forms of the example query, we had to break it into several lines.
- It can look nicer if we give names to intermediate results, and use them in subsequent steps.

Intermediate Results and Renaming

- To write out the relational algebra forms of the example query, we had to break it into several lines.
- It can look nicer if we give names to intermediate results, and use them in subsequent steps.

Intermediate Results and Renaming

- To write out the relational algebra forms of the example query, we had to break it into several lines.
- It can look nicer if we give names to intermediate results, and use them in subsequent steps.

empWithDept \leftarrow *employee* $\bowtie_{dnum=dno}$ *department*

spaceWasters \leftarrow $\sigma_{dname='Finance' \vee dname='Administration'}$ (*empWithDept*)

result \leftarrow $\pi_{fname, lname, dname}$ (*spaceWasters*)

Intermediate Results and Renaming

- To write out the relational algebra forms of the example query, we had to break it into several lines.
- It can look nicer if we give names to intermediate results, and use them in subsequent steps.

$empWithDept \leftarrow employee \bowtie_{dnum=dno} department$
 $spaceWasters \leftarrow \sigma_{dname='Finance' \vee dname='Administration'}(empWithDept)$
 $result \leftarrow \pi_{fname, lname, dname}(spaceWasters)$

You can change attribute names, relying on the *position* of values.

$renamedResult(givenName, surname, department) \leftarrow result$

Renaming operation ρ

You can also write

$$\rho_{renamedResult(givenName,surname,department)}(result)$$

for the relation *result* renamed as *renamedResult*, and with its first attribute renamed to *givenName* . . .