Relational Algebra

csc343, winter 2011
Diane Horton
University of Toronto

Why study RA?

Real databases use SQL, so why study relational algebra?

- Relational algebra is at the core of SQL
- The first thing a DBMS does with an SQL query is convert it into RA (or something very similar)
- query execution planning and query optimization are based on RA

(We'll learn SQL shortly.)

What we need to be able to express

- A database may have many tables
    - Need a way to pull out only what you want
- The information you need is often spread across several tables
    - Need a way to combine tables

RA: the general idea

- It’s an algebra, like elementary algebra in math.
- Math:
  operands like 27.5 and x
  operators like + and /
  You write expressions describing the value you want
- Relational algebra:
  operands are tables
  operators like “choose the rows that satisfy ...”
  You write expressions describing the table you want

- some operators are unary, some binary.
- operands are always relations and result is always a relation.

    R1 op R2: result is a relation

    op R: result is a relation

- So you can “compose” expressions, just like in arithmetic expressions.
- Let’s see the operations.
- (Remember, a relation/table is a *set* of tuples)

A schema for our examples (with keys underlined)

Movies(mID, title, director, year, length)

Artists(aID, aName, nationality)

Roles(mID, aID, character)

Foreign key constraints:

- Roles[mID] ⊆ Movies[mID]
- Roles[aID] ⊆ Artists[aID]

Select: choose rows

- Notation: σ_{cond} (R)
    - cond refers to the attributes of R
- The result is a relation
    - with the same schema as the operand
    - but with only the tuples satisfying the condition

- Examples:
  - All British actors
  - All movies from the 1970s
- What if we only want the names of all British actors?

Project: choose columns

- Notation: $\pi_L(R)$
  - L is a subset (not necessarily a proper subset) of the attributes of R
- The result is a relation
  - with all the tuples from R
  - but with only the attributes in L, and in that order

About project

- Why called "project"?
- What is the value of: π_{director} (Movies)
- Write an RA expression for
  - The names of all directors of movies from the 1970s
  - The names of all British actors

Duplicates

- Since relations are sets and the value of an RA expression is a relation, it does not include duplicates.
- Project can introduce duplicates that didn't exist before. How?
- DBMSs often relax the rule about duplicates. Then you need to explicitly say if you want them removed.
- Now, suppose you want the names of all characters in movies from the 1970s ...

Cartesian Product

- Notation: R x S
- The result is a relation with
  - every combination of a tuple from R1 concatenated to a tuple from R2
- It's schema is every attribute from R followed by every attribute of S, in order
- How many tuples are in R x S?
- Example: Movies x Roles
- If an attribute occurs in both relations, it occurs twice in the result (prefixed by relation name)

- Problem: many of the pairs make no sense!
- Solution?
- Back to our query: how to we get all characters from movies form the 1970s?

Cartesian product can be inconvenient

- It can introduce nonsense tuples
- You can get rid of them with selects
- But this is so highly common, an operation was defined to make it easier: join

Natural Join

- Notation: R $\bowtie$ S
- The result is formed by
  - taking the Cartesian product
  - select to ensure equality on attributes that are in both relations (determined *by name*)
  - projecting to remove duplicate attributes.
- Example: Movies $\bowtie$ Roles gets rid of the nonsense tuples

Same name, but shouldn't be forced to match

- Natural join bases the matching entirely on attribute names.
- What if two attributes have the same name, but we don't want them to have to match?
- Example: if Artists used "name" for actors' names and Movies used "name" for movies' names.
    - Can rename one of them (we'll see how)
    - Or?

Different name, but *should* be forced to match

- What if two attributes don't have the same name and we do want them to match?
- Example: Suppose we want aName and director to match
- Solution?

Building complex expressions

- Complex expressions can be composed recursively, just as in arithmetic.
- Parentheses and precedence rules define the order of evaluation.
- Precedence, from highest to lowest, is:
  σ, π, ρ
  $\times$, $\bowtie$
  $\cap$
  $\cup$, $-$
- Unless very sure, use brackets!

Breaking down complex expressions

- Complex nested expressions can be hard to read.
- Two alternative notations allow us to break them down:

  1. Expression trees.
  2. Sequences of assignment statements.

Expression Trees

- Leaves are relations.
- Interior notes are operators.
- Exactly like representing arithmetic expressions as trees.
- See section 2.4.10.

Assignment operator

- Notation: R := Expression
- Alternate notation:
  $R(A_1, ..., A_n)$:= Expression
  - Let's you name all the attributes of the new relation (not necessarily the same name they would get from Expression).
- R must be a temporary variable, not one of the relations in the schema.
  Ie, you are not updating the content of a relation!

- Example:
  - $\text{Temp1} := Q \times R$
  - $\text{Temp2} := \sigma_{a=99} (\text{Temp1}) \times S$
  - $\text{Answer(part, price)} := \pi_{b,c} (\text{Temp2})$
- Whether / how small to break things down is up to you. It's all for readability.
- As we saw, assignment can be used not only to break things down, but also to change the names of relations [and attributes].
- There is another way to rename things ...

Renaming operation

- Notation: $\rho_{R1(A1, ..., An)}$ (R2)
- The result is a relation with
  - a new relation name, R1, and
  - new attribute names, A1, ... An.
- Note that these are equivalent:
  $R1_{(A1, ..., An)}$:= R2
  R1 := $\rho_{R1(A1, ..., An)}$ (R2)
- ρ is useful if you want to rename *within* an expression.

Theta Join

- It's common to tack extra conditions on a Cartesian product.
- Theta Join makes this easier.
- Notation: $R \bowtie_{condition} S$
- The result is
  - the same as Cartesian product (not natural join!) followed by select.
- In other words, $R \bowtie_{condition} S = \sigma_{condition} (R \times S)$.

- You save just one symbol.
- You still have to write out the conditions, since they are not inferred.

Another Schema

Students($\underline{\text{sID}}$, surName, campus)

Courses($\underline{\text{cID}}$, cName, WR)

Offerings($\underline{\text{oID}}$, cID, term, instructor)

Took($\underline{\text{sID, oID}}$, grade)

Inclusion dependencies:

- Offerings[cID] ⊆ Courses[cID]
- Took[sID] ⊆ Students[sID]
- Took[oID] ⊆ Offerings[oID]

- cID is the course identifier, eg, “CSC343”
- cName is the course name, eg, “Introduction to Databases”
- WR is a boolean attribute, indicating whether or not the course satisfies the writing requirement.
- grade is a letter grade. Assume that you can compare them with >, ≥, etc.

Some queries

1. Student number of all students who have taken csc343.
2. ... and earned an A^+ in it.
3. The names of all such students.
4. The names of all students who have passed a WR course with Professor Picky.

Properties of Natural Join

- Commutative:

  $R \bowtie S = S \bowtie R$

- Associative:

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

- So writing n-ary joins is not ambiguous:

  $R_1 \bowtie R_2 \bowtie \ \ldots \ \bowtie R_3$

Special cases of natural join

No tuples match

Dept	Head
HR	Boutilier

Employee	Dept
Vista	Sales
Kagani	Production
Tzerpos	Production

Special cases of natural join

If the relations have exactly the same attributes

Artist	Name
9132	William Shatner
8762	Harrison Ford
1868	Angelina Jolie

Artist	Name
1234	Brad Pitt
1868	Angelina Jolie
5555	Patrick Stewart

Special cases of natural join

If the relations have no attributes in common

Artist	Name
1234	Brad Pitt
1868	Angelina Jolie
5555	Patrick Stewart

mID	Title	Director	Year	Length
1111	Alien	Scott	1979	152
1234	Sting	Hill	1973	130

Recall our University schema

Students(sID, surName, campus)

Courses(cID, name, WR)

Offerings(oID, cID, term, instructor)

Took(sID, oID, grade)

Inclusion dependencies:

- Offerings[cID] ⊆ Courses[cID]
- Took[sID] ⊆ Students[sID]
- Took[oID] ⊆ Offerings[oID]

An awkward query

5. sID of all students who have earned some A and some F.

Set operations

- Because relations are sets, we can use set intersection, union and difference.
- But only if the operands are relations over the same attributes (in number, name, and order).
- If the names or order mismatch?
- Now that we have these operations, let's write some more queries.

More queries

5. sID of all students who have earned some A and some F.
6. Terms when Cook and Pitassi were both teaching something.
7. Terms when either of them was teaching csc363 .
8. sID of students who have an A+ or who have passed a course taught by Atwood.
9. Terms when csc369 was not offered.
10. cID of courses that have never been offered.

For these, assume grades are numbers.

11. Names of all pairs of students who’ve taken a course together.

12. Student(s) with the highest grade in csc343, in term 20099. (Just sID for this and the rest.)

13. Students who have 100 at least twice.

14. Students who have 100 exactly twice.

15. cID of all courses that have been taught in every term when csc448 was taught.

16. Name of all students who have taken, at some point, every course Gries has taught
(but not necessarily taken them from Gries).

“Syntactic sugar”

- Some operations are not *necessary*; you can get the same effect using a combination of other operations.
- Examples: intersection, theta join
- We call this “syntactic sugar”
- This concept also comes up in logic and programming languages.

Summary of operators

Operation	Name	Symbol
choose rows	select	σ
choose columns	project	π
combine tables	Cartesian product	$\times$
	natural join	$\bowtie$
	theta join	$\bowtie_{condition}$
set operations	intersection, union, subtraction	$\cap$ $\cup$ $-$
rename relation [and attributes]	rename	ρ
assignment	assignment	:=

RA is procedural

- An RA query itself describes a procedure for constructing the result (*i.e.*, how to implement the query).
- We say that it is "procedural."

Evaluating queries

- Although an RA expression suggests a query execution plan, many other expressions would yield the same result.
- Some of the alternatives may suggest a more efficient query execution plan.
- Which is best depends also on the data in the database.
- In real DBMSs, the query is optimized (rewritten in a more efficient form) before evaluation.
- This is a topic in csc443.

Relational Calculus

- Another abstract query language for the relational model.
- Based on first-order logic.
- RC is "declarative": the query describes what you want, but not how to get it.
- Queries look like this:
  $\{ t \mid t \in \text{Movies} \wedge t[\text{director}] = \text{“Scott”} \}$
- Expressive power (when limited to queries that generate finite results) is same as RA. It is "relationally complete."

Tips and Tricks for RA

- Ask yourself which relations need to be involved.
- Break the answer down and use good names for intermediate results.
  Add a comment the explains exactly what it contains.
- Annotate each subexpression, to show the attributes of its resulting relation.
- Every time you combine relations, confirm that (a) attributes that should match will be made to match and (b) attributes that will be made to match should match.

Tips and Tricks for RA

- Is there an intermediate relation that would help you get the final answer? Draw it out with actual data in it.

Tricks for specific types of query

- Max (min is analogous): Pair tuples and find those that are *not* the max. Then substract from all to find the max[es].
- To show "at least k": Make all combos of k different tuples that meet the required condition.
- To show "exactly k":
    - Show "k or more".
    - Then subtract "(k+1) or more".

- To show “every”:
    - Make all combos that should have occurred.
    - Subtract those that *did* occur to find those that didn’t always. These are the failures.
    - Subtract the failures from all to get the answer.