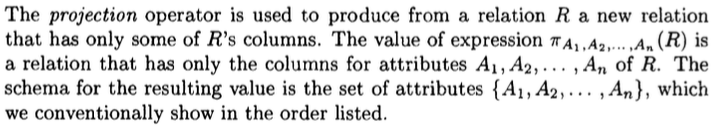
* [2.4] Relational Algebra
* Language for querying a relational database.
  + Basis for SQL.
* Why?
  + Less powerful than C++/Java is a good thing!
  + Easy to learn
  + Easy for DBMS to optimize
  + Why better than SQL – often there's a 1-1 correspondence between SQL queries and rel'n algebra queries, but the reln algebra is shorter bc you use symbols.
* Relational algebra is similar to algebra from math, and to Boolean algebra
  + In algebra, you operate on numbers. Boolean alg, operate on T/F
    - In relational algebra, operate on RELATIONS
  + Operations in algebra are +, -, \*, /….boolean alg= 'and' 'or' 'not'
    - In relational algebra, we have operations that are appropriate to relations.
* Set operations on relations.
  + Assume we have two relations, called R and S.
  + Assume R and S **have identical schemas** (same attribs/columns, and the domains/types match). Obv R and S don’t have to have the same rows.
  + R U S = union R \cap S = intersection. R – S = difference
* Examples
  + What is Students union Professors? [8 rows, 2 cols: pot, gra, wea, long, mal, mcg, sna, dum]
    - Meaning of this query?
  + Students intersect Professors? [1 row, 2 cols: long]
    - Meaning?
  + Gryffindors – Students = [2 rows, 2 cols: mcg, dum]
    - Meaning?
  + Stu – Gryf = [1 row,2 cols: mal]
* Projection (pi)
  + Projection operator takes a relation and produces a new relation with only certain columns.
  + 
  + ­Examples from NEW student database (not the harry potter one):
  + pi\_name(Students) – returns all six students
  + pi\_major(Students) – returns 3 rows [cs, math, physics]
  + pi\_dept(courses) – returns 4 depts.
  + Pi\_dept,seats(courses) – returns how many rows? [cs20, cs15, cs25, math18, math20, music10, phys16, music21, phys15, math20] (math20 is twice, only appears once, so 9 rows).
* Selection operator (sigma)
  + Selection allows us to select only certain rows from a relation.
  + We are allowed to specify a condition that various attributes of the rows must satisfy.
  + Sigma\_[major = CS] (Students)
  + Sigma\_[depth=music] (Courses)
  + Sigma\_[major=CS AND age <= 19] (Students) Alice, Carol
    - How could we do this without an AND? [use intersection]
* Rename operator.
  + Used for renaming a relation and/or its columns.
  + Rho\_[S(A1, A2…)] ( R) has exact same rows and columns as R, they are just renamed to A1, A2… and the relation itself is renamed to S.
  + Can also use Rho[S]( R) to just rename the relation but keep the rows the same.
* Combining/chaining operators
  + Give me a list of the names of the CS majors.
    - Pi[name]sigma[major=CS](S)
  + Give me the crn&names of all math or CS courses with < 20 seats]
    - Pi[crn,name]sigma[(dept=CS or dept=math) and seats<20](C)

COMBINING MULT RELATIONS

* Cartesian product.
* Give def first => R x S, takes 2 relations and pairs each row in the left table with every possible row in the 2nd table. So the resulting relation always has (# rows in first table) x (# rows 2nd table)
* Show slide example.
  + We want to get a list of all possible partners of math people with CS people.
  + Get math people = pi[name]sigma[major=math](S) [bob,eva]
  + Get CS people = pi[name]sigma[major=CS](S) [alice, carol, dan]
  + Cross them, but what are the names of the columns? They are identical! Argh! Let’s rename them.
  + Rho[S(mathname)] ( above piece ) X Rho[S(csname)](above piece)
  + Yields => 2 column table, (bob alice) (bob carol) (bob dan) (eva alice) (eva carol) (eva dan)
* Cartesian product is useful, but it combines two tables without considering content of the rows. More often we want to combine two tables based on examining the content of one or more attributes, and pick only the rows that satisfy some condition on those attributes. This is called a JOIN.
* Natural Join **[NEEDS A BETTER EXAMPLE WITH SYNTHETIC DATA, PROB]**
  + A natural join of two relations, R  S, is just like a cross product except it only matches rows between the two relations where ***all common attributes are identical***.
  + So a natural join is a three step process.
    - Step 1 – identify attributes that both relations share (same name)
    - Step 2 – create a new relation with all attributes from both relations (one copy of the shared attributes).
    - Step 3 – add rows to the relation for every pair of rows from R and S, but only where the attribs match.
  + Show slide example.
  + A natural join is most often used when you have information in one table that refers to information in another table (usually through a connection through a key in one of the tables, though not always). Often you use this type of join to add the info from the second table into the first one.
  + Ex: natural join Students and Enrolled.
    - Name, ID, Major, Age, CRN.
  + Ex: nat join Courses and Enrolled.
    - CRN, Dept, Name, Seats, ID.
  + Can I nat join all three? Yes!
  + Show slide caveats.
  + As a class
  + I want a list of the CRNS Alice is enrolled in.
    - Pi[CRN]Sigma[name=Alice](Students NATJOIN Enrolled)
    - OR
    - Pi[CRN] [ Sigma[name=alice](S) NATJOIN Enrolled ]
    - Aside for query optimization. Many equivalent expressions. The query optimizer in a DBMS is designed to pick which one of these will operate most efficiently.
  + I want a list of the CourseNames Alice is enrolled in.
    - Prev query -> nat join to Courses,
    - Then Pi[coursename]
* Theta-join
  + Theta-join is more flexible than a natural join.
  + Allows combining tables on any condition of the attributes you specify.
  + Example – give me a list of students, with their majors, and all the courses taught in the same dept as their major.
    - Students THETAJOIN[S.major=C.dept] Courses.
  + Theta joins often come up in joining a table to itself.
    - Want a list of all possible pairs of CS partners.
    - Get all CS majors.
    - What happens with cross product? [people paired with themselves]
    - Theta-join on not equal.
    - Theta-join on less than.
* Equivalences
  + R thetajoin[C] S = sigma[C](R x S)
  + Can also write nat join in terms of cross product.
* Linear notation (shorthand notation) [ allows you to give temporary names to relations as you build a query]
  + Give me list of the CourseNames Alice is enrolled in. [this is the example from above]
* Harder Practice.
  + Coursenames of courses that have somebody taking that course.
    - Pi[Coursename] (Enroll NATJOIN Courses)
  + Coursenames that are empty.
    - Pi[Coursename](Courses) – previous query.
  + Students with the courses they are taking that are out of their dept. (studname, coursename)
    - Pi[studname, coursename] Sigma[Major != Dept] (Students NJ Enrolled NJ Courses)

|  |  |
| --- | --- |
| a. | Find all pizzerias frequented by at least one person under the age of 18. |
| b. | Find the names of all Rhodes students who eat either mushroom or pepperoni pizza (or both). |
| c. | Find the names of all Rhodes students who eat both mushroom and pepperoni pizza. |
| d. | Find all pizzerias that serve at least one pizza that Amy eats for less than $10.00. |
| e. | Find all pizzerias that are frequented by only Rhodes Students or only U of M students. |
| f. | For each person, find all pizzas the person eats that are not served by any pizzeria the person frequents.  Return all such person (name) / pizza pairs. |
| g. | Find the names of all people who frequent only pizzerias serving at least one pizza they eat. |
| h. | Find the names of all people who frequent every pizzeria serving at least one pizza they eat. |
| i. | Find the pizzeria serving the cheapest pepperoni pizza. In the case of ties, return all of the cheapest-pepperoni pizzerias. |

