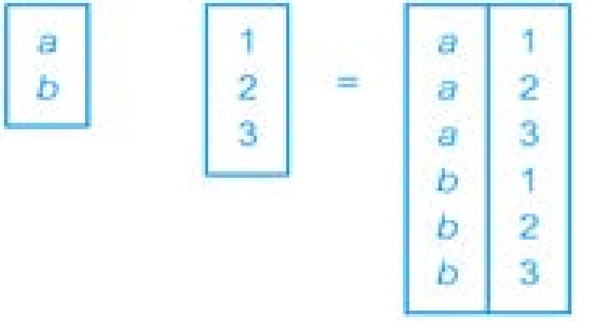
Fundamental Operators: Binary

- Cartesian Product: R X S
  - Returns a new relation that is the concatenation of each tuple in relation R with each tuple in relation S
  - Resulting union relation will be of cardinality:
    - |R|*|S|
  - Attributes with same name in R and S should be tagged with Relation ID (R,S) to differentiate



Fundamental Operators: Cartesian Product

Course X Enrollment

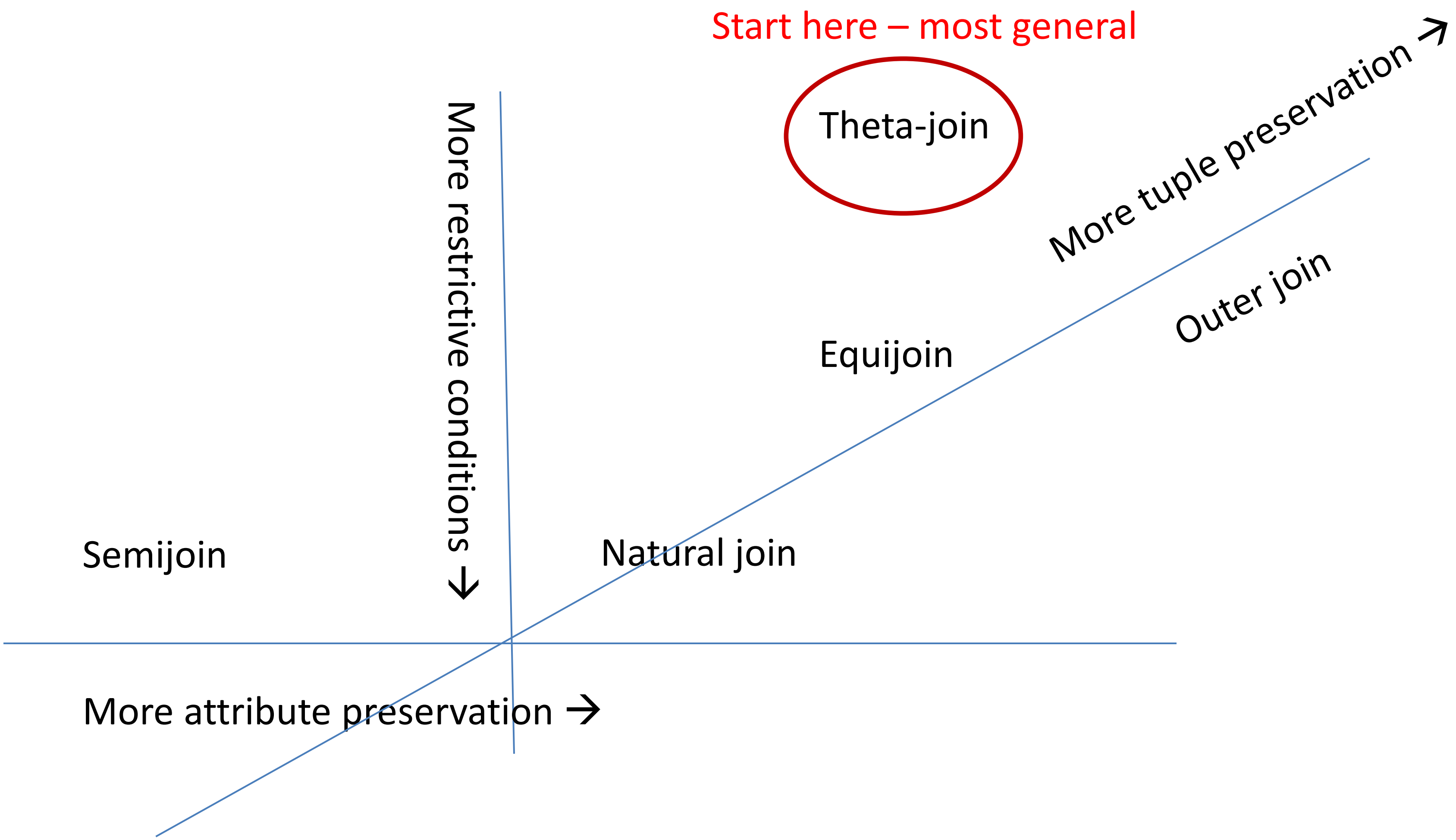
Course ID	dept.	number	student ID	Course ID
06902	CSC	221	1123	06902
06903	CSC	231	1123	06902
06904	CSC	241	1123	06902
06902	CSC	221	1129	06902
06903	CSC	231	1129	06902
06904	CSC	241	1129	06902
06902	CSC	221	1145	06902
06903	CSC	231	1145	06902
06904	CSC	241	1145	06902
06902	CSC	221	1123	06903
06903	CSC	231	1123	06903
06904	CSC	241	1123	06903

courseID	dept.	number	student ID	Course ID
06902	CSC	221	1145	06903
06903	CSC	231	1145	06903
06904	CSC	241	1145	06903
06902	CSC	221	1123	06904
06903	CSC	231	1123	06904
06904	CSC	241	1123	06904
06902	CSC	221	1129	06904
06903	CSC	231	1129	06904
06904	CSC	241	1129	06904

courseID	dept.	number	student ID	Course ID
06905	CSC	191	1123	06902
06905	CSC	191	1129	06902
06905	CSC	191	1145	06902
06905	CSC	191	1123	06903
06905	CSC	191	1145	06903
06905	CSC	191	1123	06904
06905	CSC	191	1129	06904

Forgive the differences in capitalization
and the weird ordering
Also, the courseIDs would need to be renamed
One way, tag with relation they were in

Joins in 3-dimensions



Higher Order Operators: Join Operators

- Joins: Employed when want to create new relations with data from two relations, but only want those tuples containing certain properties

- Essentially all joins are fundamentally:
  - Cartesian product, followed by Selection
    $R3 = \sigma_p(R1 \; X \; R2)$

- Multiple different applications lead to definition of several special variations of join operators

Higher Order Operators: Join Operators

- Theta-join (θ-join): $R \bowtie_F S$
  - Returns a new relation that contains the tuples out of the Cartesian product of R and S satisfying predicate F, where F is defined over comparisons R.a θ S.b and θ is limited to { <, >, <=, >=, =, !=)

- Equivalent to σ_F (R X S)

- Note resulting relation could be empty, if predicate not satisfied by any tuple

Higher Order Operators: Theta-Join

- θ -join example:

Assume have a table of "Honors" based on GPA, want list of students and their possible Honors

Student

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

Honors

Name	gpaReq
Dean's List	3.0
President's List	3.5
Superstar	4.0

$$\text{Student} \bowtie_{\text{Student.GPA} > \text{Honors.gpaReq}} \text{Honors}$$

Higher Order Operators: Theta-Join

Cartesian Product – look for GPA > gpaReq

Student | | | | | | Honors |
---|---|---|---|---|---|---|---
| Student ID | last Name | first Name | year | major | GPA | name | gpaReq |
| 1123 | Smith | Robert | 4 | CSC | 3.5 | Dean's List | 3.0 |
| 1123 | Smith | Robert | 4 | CSC | 3.5 | President's List | 3.5 |
| 1123 | Smith | Robert | 4 | CSC | 3.5 | Superstar | 4.0 |
| 1129 | Jones | Douglas | 3 | MTH | 2.9 | Dean's List | 3.0 |
| 1129 | Jones | Douglas | 3 | MTH | 2.9 | President's List | 3.5 |
| 1129 | Jones | Douglas | 3 | MTH | 2.9 | Superstar | 4.0 |
| 1145 | Brady | Susan | 4 | CSC | 3.8 | Dean's List | 3.0 |
| 1145 | Brady | Susan | 4 | CSC | 3.8 | President's List | 3.5 |
| 1145 | Brady | Susan | 4 | CSC | 3.8 | Superstar | 4.0 |

Higher Order Operators: Theta-Join

Output Relation

Student ID	last Name	first Name	year	major	GPA	name	gpaReq
1123	Smith	Robert	4	CSC	3.5	Dean's List	3.0
1145	Brady	Susan	4	CSC	3.8	Dean's List	3.0
1145	Brady	Susan	4	CSC	3.8	President's List	3.5

Think about the Cartesian product that we would have seen before the selection was employed

Higher Order Operators: θ-join Specialization

- There are two simple specializations of θ-join:
  - If the operator in the predicate F is constrained to equals (=), call it an *equijoin*
  - If it is an equijoin, and the predicate only compares same name attribute(s), call it a *natural join R* $\bowtie$ *S*.
    - Also drop one instance of each duplicate column (via a projection operator)

If a tuple in relation R doesn't have a matching value in the same-name attributesin relation S, it <u>does not </u>appear in the output relation.

T

A	B
a	1
b	2

U

B	C
1	x
1	y
3	z

T $\bowtie$ *U*

A	B	C
a	1	x
a	1	y

Higher Order Operators: θ-join Specialization

- Equijoin example: $Course \bowtie_{Course.courseID = Enrollment.courseID} Enrollment$

Course. Course ID	dept.	number	student ID	Enroll ment. Course ID
06902	CSC	221	1123	06902
06902	CSC	221	1129	06902
06902	CSC	221	1145	06902
06903	CSC	231	1123	06903
06903	CSC	231	1145	06903
06904	CSC	241	1123	06904
06904	CSC	241	1129	06904

What is the difference between these two?

- Natural join example:

Course. Course ID	dept.	number	student ID
06902	CSC	221	1123
06902	CSC	221	1129
06902	CSC	221	1145
06903	CSC	231	1123
06903	CSC	231	1145
06904	CSC	241	1123
06904	CSC	241	1129

$Course \bowtie Enrollment$

Note these do not contain references to 06905 CSC 191, which no one is enrolled in

Higher Order Operators: Semijoin

- The semijoin $\ltimes$ of R and S is essentially a θ-join, followed by a projection to only the attributes of R
  - $\Pi_{col1,\,...,\,coln}(\ \mathbf{R} \bowtie_{\mathbf{F}} \mathbf{S}\ )$, *where col1..coln are columns of R* $\Rightarrow$ $\Pi_{col1,\,...,\,coln}(\sigma_F(\mathrm{R} \times \mathrm{S}))$
  - Naturally extends to equijoins and natural joins

Higher Order Operators: Semijoins

- Semijoin examples: Course $\ltimes$ Enrollment

Take result from join
(this is a natural join)

Course. Course ID	dept.	number	student ID
06902	CSC	221	1123
06902	CSC	221	1129
06902	CSC	221	1145
06903	CSC	231	1123
06903	CSC	231	1145
06904	CSC	241	1123
06904	CSC	241	1129

Project to Course attributes
Reduces attributes, drops duplicates

courseID	dept.	number
06902	CSC	221
06903	CSC	231
06904	CSC	241

This purely gives course information for those courses for which someone is enrolled

Higher Order Operators: Tuple-Preserving Joins

- Joins in which we preserve tuples from one, other, or both of the input relations even if they don't match
  - *Outer joins*: extensions of natural join
    - Left outer join: $⟕$ preserve tuples from left-hand relation w/o a match
    - Right outer join: $⟖$ preserve tuples from right-hand relation w/o a match
    - Full outer join: $⟗$ preserve tuples from both relations w/o a match
  - Use NULL to pad

Higher Order Operators: Tuple-Preserving Joins

- Outer join examples: Give me a list of all courses, annotated with ids of students enrolled in those courses
  - Want to include zero-enrollment classes, as the Dean may want to talk to Department about why offered a class that no-one signed up for → use Left (natural) outer join

(Course ⟕ Enrollment)

Higher Order Operators: Tuple-Preserving Joins

(Course ⟕ Enrollment)

Course. Course ID	dept.	number	student ID
06902	CSC	221	1123
06902	CSC	221	1129
06902	CSC	221	1145
06903	CSC	231	1123
06903	CSC	231	1145
06904	CSC	241	1123
06904	CSC	241	1129
06905	CSC	191	NULL

Higher Order Operators: Division

- The division operator, R / S, is defined as follows:
  - *Assumptions:* Assume R has attribute set A and S has attribute set B and B is a subset of A. The attributes unique to A are C.
  - The division operator returns a new relation over attributes in C for which tuples in R match the combination of **every** tuple in S.
  - Simplify to think about:
    - Assume R has X,Y attributes; S has attribute Y
      - Looking across tuples of S, might have multiple values for Y, such as {1,2,8}
      - R / S is going to give you back any values X such that in R an X value is paired with all values of Y that showed up in S (so any values for X from R which were paired with 1,2, AND 8 in R)
    - Typically: Show me those entities X involved in all things Y
      - Which suppliers can produce all these parts?
      - Which students took all of these courses?

Higher Order Operators: Division

- Division example: Enrollment / $\Pi_{studentID}$Student

Enrollment

studentID	courseID
1123	06902
1129	06902
1145	06902
1123	06903
1145	06903
1123	06904
1129	06904

Student

studentID
1123
1129
1145

Enrollment / $\Pi_{studentID}$Student

courseID
06902

The courses that enroll all students

Higher Order Operators: Division

- How do we get at “the students enrolled in all courses”, and are there any?

Higher Order Operators: Division

- Division example: Enrollment / $\Pi_{courseID}$ *Course*

Enrollment

studentID	courseID
1123	06902
1129	06902
1145	06902
1123	06903
1145	06903
1123	06904
1129	06904

Enrollment / $\Pi_{courseID}$Course

studentID

The students enrolled in all courses

Course

courseID	dept.	number
06902	CSC	221
06903	CSC	231
06904	CSC	241
06905	CSC	191

Relational Algebra Operators

- Note that four of the relational algebra operators had a corollary directly in set operators:
  - Union, Intersection, Difference, Cartesian Product
- Other four operators are unique to relational algebra:
  - Selection, Projection, Join, Division

Schema Changes

- Which operators provide relations with new schemas (structures) from the inputs?
  - These do not:
    - Intersection, Union, Difference, Selection
  - These do:
    - Projection, Cartesian product, Join, Rename

Operator Precedence

- Given a sequence of operators, the following precedence rules hold:
    1. [σ, π, rename] (highest)

       -- Select, Project, Rename
    2. [X, ⋈] – Cartesian product, Join
    3. ∩ -- Intersection
    4. [∪, —] – Union, Difference

Practice: Hotel Schema

Hotel

<u>hotelNumber</u>	hotelName	city

Room

<u>roomNumber</u>	<u>hotelNumber</u>	type	price

Guest

<u>guestNumber</u>	guestName	guestAddress

Booking

<u>hotelNumber</u>	<u>guestNumber</u>	<u>dateFrom</u>	dateTo	roomNumber

Practice: Hotel Schema: Foreign Keys

Hotel

<u>hotelNumber</u>	hotelName	city

Room

<u>roomNumber</u>	<u>*hotelNumber*</u>	type	price

Guest

<u>guestNumber</u>	guestName	guestAddress

Booking

<u>*hotelNumber*</u>	<u>*guestNumber*</u>	<u>dateFrom</u>	dateTo	*roomNumber*

Practice: Hotel Schema

Provide both a physical description and an English description of the relations that would be produced by the following relational algebra operations

a. $\Pi_{hotelNumber}(\sigma_{price > 50}(Room))$
b. $\sigma_{Hotel.hotelNumber=Room.hotelNumber}(Hotel \times Room)$
c. $\Pi_{hotelName}(Hotel \bowtie (\sigma_{price > 50}(Room)))$
d. Guest LOJ $(\sigma_{dateTo >= \text{'1-Jan-2007'}}(Booking))$
e. Hotel $\rhd_{Hotel.hotelNumber=Room.hotelNumber}(\sigma_{price > 50}(Room))$
f. $\Pi_{guestName,hotelNumber}(Booking \bowtie_{Booking.guestNumber=Guest.guestNumber} Guest) \;/\; \Pi_{hotelNumber}(\sigma_{city=\text{'London'}}(Hotel))$

Practice: Hotel Schema

a. $\Pi_{hotelNumber}(\sigma_{price > 50}(Room))$
Physical: One attribute table (hotelNumber)
English: Show room numbers of all hotels with price > $50

b. $\sigma_{Hotel.hotelNumber=Room.hotelNumber}(Hotel \times Room)$
Physical: All attributes from both Hotel and Room, including duplicate attributes
English: Show all rooms in all hotels

c. $\Pi_{hotelName}(Hotel \bowtie (\sigma_{price > 50}(Room)))$
Physical: One attribute table (hotelNames)
English: Show all hotel names for hotels that have rooms priced > $50

Practice: Hotel Schema

d. Guest LOJ ($\sigma_{dateTo >= \text{'1-Jan-2007'}}$(Booking))
Physical: All guest attributes and all booking attributes, but only one copy of guestNumber
English: Show all guests and their information, and include any information about their bookings from 1-Jan-2007

e. Hotel $\rhd_{Hotel.hotelNumber=Room.hotelNumber}$($\sigma_{price > 50}$(Room))
Physical: Only hotel attributes
English: Show all hotel details for any hotels that have rooms priced over $50

f. $\Pi_{guestName,hotelNumber}$(Booking $\rhd\lhd_{Booking.guestNumber=Guest.guestNumber}$ Guest) / $\Pi_{hotelNumber}$($\sigma_{city=\text{'London'}}$(Hotel))
Physical: One attribute table, guestName
English: Show all guests who have booked rooms in **all** London hotels

Practice: Hotel Schema

- Generate correct (and reasonable, given assumptions you may need to make about the intent of the query) relational algebra expressions for the following queries:
  - List all hotels.
  - List all single rooms with a price below $100 a night.
  - List the names and cities of all guests.
  - List the price and type of all rooms at the Winston hotel.
  - List all guests currently staying at the Winston hotel.
  - List the details of all rooms at the Winston hotel, including the name of the guest staying in the room if the room is occupied.
  - List the guest details of all guests staying at the Winston hotel.

Practice: Hotel Schema

– List all hotels.

RA: Hotel

– List all single rooms with a price below \$100 a night.

RA: $\sigma_{type='single' \;\&\&\; price<\$100}$(Room)

– List the names and cities of all guests.

RA: $\Pi_{guestName,guestAddress}$Guest)

– List the price and type of all rooms at the Winston hotel.

RA: $\Pi_{price,type}$(Room $\rhd_{Room.hotelNumber=Hotel.hotelNumber}$ ($\sigma_{hotelName='Winston'}$ (Hotel)))