Relational Algebra 1

Week 4

Relational Query Languages

- *Query languages:* Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic.
  - Allows for much optimization.
- Query Languages != programming languages!
  - QLs not expected to be "Turing complete".
  - QLs not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

Formal Relational Query Languages

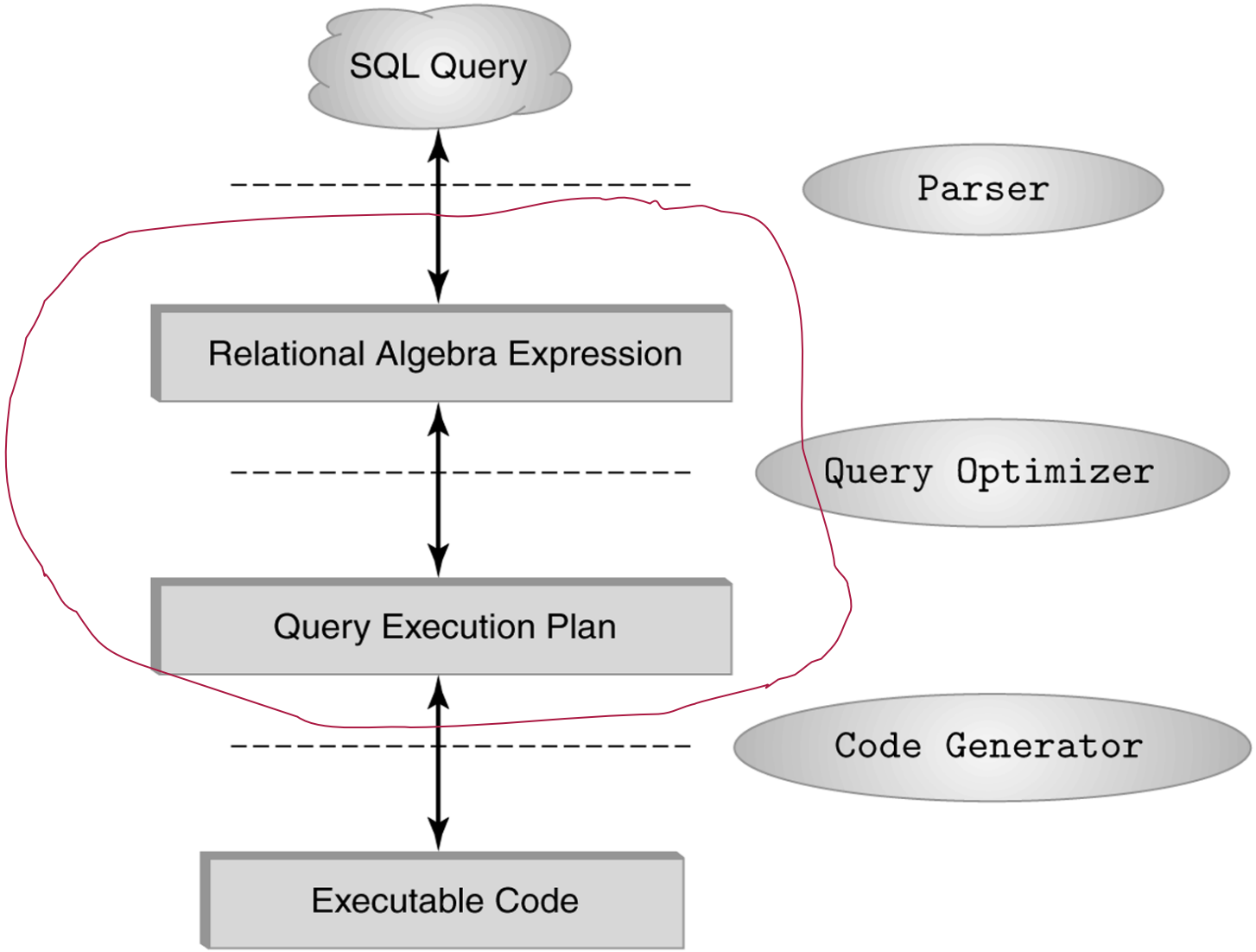
Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:

❶ *Relational Algebra*: More operational, very useful for representing execution plans.

❷ *Relational Calculus*: Lets users describe what they want, rather than how to compute it. (Non-operational, *declarative*.)

☛ *Understanding Algebra is key to understanding SQL, and query processing!*

The Role of Relational Algebra in a DBMS



Algebra Preliminaries

- A query is applied to *relation instances*, and the result of a query is also a relation instance.
  - *Schemas* of input relations for a query are fixed (but query will run regardless of instance!)
  - The schema for the *result* of a given query is also fixed! Determined by definition of query language constructs.

Relational Algebra

- Procedural language
- Five basic operators
  - **selection** — select
  - **projection** — project
  - **union** — (why no intersection?)
  - **set difference** — difference
  - **Cross product** — Cartesian product

SQL is closely based on relational algebra.

- The are some other operators which are composed of the above operators. These show up so often that we give them special names.
- The operators take one or two relations as inputs and give a new relation as a result.

Select Operation – Example

- Relation *r*

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A=B \wedge D > 5}(r)$

lowercase Greek sigma

A	B	C	D
α	α	1	7
β	β	23	10

Intuition: The **select** operation allows us to retrieve some rows of a relation (by "some" I mean anywhere from none of them to all of them)

Here I have retrieved all the rows of the relation *r* where the value in field *A* equals the value in field *B*, and the value in field *D* is greater than 5.

Select Operation

- Notation: $\sigma_p(r)$ lowercase Greek sigma σ
- p is called the **selection** predicate
- Defined as:

$$\sigma_p(\boldsymbol{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

Where p is a formula in propositional calculus consisting of terms connected by : ∧ (**and**), ∨ (**or**), ¬ (**not**)
Each term is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: =, ≠, >, ≥, <, ≤

- Example of selection:

$\sigma_{name = \text{'Lee'}}(professor)$

Project Operation – Example I

- Relation *r*:

A	B	C
α	10	7
α	20	1
β	30	1
β	40	2

- $\pi_{A,C}\ (r)$

Greek lower-case pi

A	C
α	7
α	1
β	1
β	2

Intuition: The **project** operation allows us to retrieve some columns of a relation (by "some" I mean anywhere from none of them to all of them)

Here I have retrieved columns *A* and *C*.

Project Operation – Example II

- Relation r:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- $\pi_{A,C}\,(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Intuition: The project operation removes duplicate rows, since relations are sets.

Here there are two rows with $A = \alpha$ and $C = 1$. So one was discarded.

Project Operation

- Notation:

$$\pi_{A1,\, A2,\, \ldots,\, Ak}\,(r)$$

Greek lower-case pi

where A_1, A_2 are attribute names and r is a relation name.

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets.

Union Operation – Example

Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

r ∪ s:

A	B
α	1
α	2
β	1
β	3

Intuition: The **union** operation concatenates two relations, and removes duplicate rows (since relations are sets).

Here there are two rows with *A* = α and *B* = *2*. So one was discarded.

Union Operation

- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

"Union-compatible"

For $r \cup s$ to be valid.

1. *r, s* must have the *same arity* (same number of attributes)
2. The attribute domains must be *compatible* (e.g., 2nd column of *r* deals with the same type of values as does the 2nd column of *s*).

Although the field types must be the same, the names can be different. For example I can union *professor* and *lecturer* where:

professor(<u>*PID*</u> : string, *name* : string)
lecturer(<u>*LID*</u> : string, *first_name* : string)

Related Operation: Intersection

Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

r ∩ s:

A	B
α	2

- Similar to Union operation.
- But Intersection is NOT one of the five basic operations.
- Intuition: The **intersection** operation computes the common rows between two relations

Set Difference Operation – Example

Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

r – *s*:

A	B
α	1
β	1

Intuition: The **set difference** operation returns all the rows that are in *r* but not in *s*.

Set Difference Operation

- Notation $r - s$
- Defined as:

$$r - s = \{t \mid t \in r \textbf{ and } \text{t} \notin s\}$$

- Set differences must be taken between *compatible* relations. "Union-compatible"
  - r and s must have the *same arity*
  - attribute domains of r and s must be compatible
- Note that in general $r - s \neq s - r$

Cross-Product Operation-Example

Relations *r, s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

r x *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Intuition: The **cross product** operation returns all possible combinations of rows in *r* with rows in *s*.

In other words the result is every possible pairing of the rows of *r* and *s*.

Cross-Product Operation

- Notation r x s
- Defined as:

$$r \text{ x } s = \{t\, q \mid t \in r \textbf{ and } q \in s\}$$

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \varnothing$).
- If attributes names of *r(R)* and *s(S)* are not disjoint, then renaming must be used.

Composition of Operations

- We can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

r x *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

"take the cross product of *r* and *s*, then return only the rows where *A* equals *B*"

$\sigma_{A=C}(r \times s)$ ➡

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.

Example:

$\rho\ (myRelation,\ (r - s))$

Renaming columns (rename A to A2):

$\rho\ (myRelation(A->A2),\ (r - s))$

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

S

Take the set difference of r and s, and call the result myRelation
Renaming in relational algebra is essentiality the same as assignment in a programming language

A	B
α	1
β	1

myRelation

Rename Operation

If a relational-algebra expression Y has arity n, then

$$\rho(X(A->A1,\ B->A2,\ \ldots),\ Y)$$

returns the result of expression Y under the name X, and with the attributes renamed to $A1,\ A2,\ \ldots.,\ An$.

For example,

$\rho\ (myRelation(A->E,\ B->K),\ (r - s))$

Take the set difference of r and s, and call the result myRelation, while renaming the first field to E, and the second field to *K*.

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

S

E	K
α	1
β	1

myRelation

Sailors Example

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

Example Instances

- "Sailors" and "Reserves" relations for our examples.

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Algebra Operations

- Look what we want to get from the following table:

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Selection

- Selects rows that satisfy *selection condition*.
- No duplicates in result! (Why?)
- *Schema* of result identical to schema of (only) input relation.

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$$\sigma_{rating>8}(S2) =$$

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

Projection

- Deletes attributes that are not in *projection list*.
- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate *duplicates*! (Why??)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it.

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$$\pi_{sname,rating}(S2)$$

age
35.0
55.5

$$\pi_{age}(S2)$$

Composition of Operations

- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition*)

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$$\pi_{sname,rating}(\sigma_{rating>8}(S2))=$$

sname	rating
yuppy	9
rusty	10

What do we want to get from two relations?

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

What about: Who reserved boat 101?

Or: Find the name of the sailor who reserved boat 101.

Cross-Product

- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names inherited.

sid1	sname	rating	age	sid2	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

☛ *Renaming operator (because of naming conflict)*:

$$\rho(sid \rightarrow sid1,\ S1) \times \rho(sid \rightarrow sid2,\ R1)$$

Why does this cross product help

Query: Find the name of the sailor who reserved boat 101.

$$Temp = \rho(sid \rightarrow sid1, S1) \times \rho(sid \rightarrow sid2, R1)$$

$$\text{Result} = \pi_{Sname}(\sigma_{sid1 = sid2 \wedge bid = 101}(Temp))$$

* Note my use of "temporary" relation Temp.

Another example

- Find the name of the sailor having the highest rating.

$$\text{AllR} = \pi_{ratingA} \rho(rating \text{->} ratingA, S2)$$

$$\text{Result?} = \pi_{Sname}(\sigma_{rating<ratingA}(S2 \times \text{AllR}))$$

What's in "Result?" ?

Does it answer our query?

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

×

AllR

ratingA
9
8
5
10

=

sid	sname	rating	age	ratingA
28	yuppy	9	35.0	9
28	yuppy	9	35.0	8
28	yuppy	9	35.0	5
28	**yuppy**	**9**	**35.0**	**10**
31	**lubber**	**8**	**55.5**	**9**
31	lubber	8	55.5	8
31	lubber	8	55.5	5
31	**lubber**	**8**	**55.5**	**10**
44	**guppy**	**5**	**35.0**	**9**
44	**guppy**	**5**	**35.0**	**8**
44	guppy	5	35.0	5
44	**guppy**	**5**	**35.0**	**10**
58	rusty	10	35.0	9
58	rusty	10	35.0	8
58	rusty	10	35.0	5
58	rusty	10	35.0	10

$$\text{AllR} = \pi_{ratingA}\, \rho(rating\text{->}ratingA, S2)$$

$$\text{Result?} = \pi_{Sname}(\sigma_{rating<ratingA}(S2 \times \text{AllR}))$$

Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - 'Corresponding' fields have the same type.
- What is the *schema* of result?

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

Back to our query

- Find the name of the sailor having the highest rating.

$$\text{AllR} = \pi_{ratingA}\, \rho(rating \text{->} ratingA,\ S2)$$

$$\text{Tmp} = \pi_{Sid,Sname}(\sigma_{rating < ratingA}(S2 \times \text{AllR}))$$

$$\text{Result} = \pi_{Sname}\,(\pi_{Sid,Sname}(S2) - \text{Tmp})$$

* Why not project on Sid only for Tmp?

Relational Algebra (So far)

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* (×) Allows us to combine two relations.
  - *Set-difference* (-) Tuples in reln. 1, but not in reln. 2.
  - *Union* (∪) Tuples in reln. 1 and tuples in reln. 2.

  Also,
  - *Rename* (ρ) Changes names of the attributes
  - *Intersection* (∩) Tuples in both reln. 1 and in reln. 2.

- Since each operation returns a relation, operations can be *composed*! (Algebra is "closed".)
- Use of temporary relations recommended.

Additional Operations

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

- Natural join
- Conditional Join
- Equi-Join
- Division

All joins are really special cases of conditional join

Also, we 've already seen "Set intersection":

$$r \cap s = r - (r - s)$$

Quick note on notation

good_customers

customer-name	***loan-number***
Patty	1234
Apu	3421
Selma	2342
Ned	4531

bad_customers

customer-name	***loan-number***
Seymour	3432
Marge	3467
Selma	7625
Abraham	3597

If we have two or more relations which feature the same attribute names, we could confuse them. To prevent this we can use dot notation. For example

good_customers.loan-number

Natural-Join Operation: Motivation

Very often, we have a query and the answer is not contained in a single relation. For example, I might wish to know where Apu banks.
The classic relational algebra way to do such queries is a cross product, followed by a selection which tests for equality on some pair of fields.

$\sigma_{borrower.l\text{-}number = loan.l\text{-}number}(borrower \times loan)))$

While this works…

- it is unintuitive
- it requires a lot of memory
- the notation is cumbersome

borrower

cust-name	***l-number***
Patty	1234
Apu	3421

loan

l-number	***branch***
1234	Dublin
3421	Irvine

cust-name	***borrower.l-number***	***loan.l-number***	***branch***
Patty	1234	1234	Dublin
Patty	1234	3421	Irvine
Apu	3421	1234	Dublin
Apu	3421	3421	Irvine

cust-name	***borrower.l-number***	***loan.l-number***	***branch***
Patty	1234	1234	Dublin
Apu	3421	3421	Irvine

Note that in this example the two relations are the same size (2 by 2), this does not have to be the case.

So, we have a more intuitive way of achieving the same effect, the natural join, denoted by the $\bowtie$ symbol

Natural-Join Operation: Intuition

Natural join combines a cross product and a selection into one operation. It performs a selection forcing equality on *those attributes that appear in both relation schemes*. Duplicates are removed as in all relation operations.

So, if the relations have one attribute in common, as in the last slide ("*l-number*"), for example, we have…

$$borrower \bowtie loan = \sigma_{borrower.l\text{-}number = loan.l\text{-}number}(borrower \times loan)))$$

There are two special cases:

- If the two relations have no attributes in common, then their natural join is simply their cross product.
- If the two relations have more than one attribute in common, then the natural join selects only the rows where all pairs of matching attributes match. (let' s see an example on the next slide).

A

l-name	*f-name*	*age*
Bouvier	Selma	40
Bouvier	Patty	40
Smith	Maggie	2

B

l-name	*f-name*	*ID*
Bouvier	Selma	1232
Smith	Selma	4423

Both the *l-name* and the *f-name* match, so select.

Only the *f-names* match, so don't select.

Only the *l-names* match, so don't select.

l-name	*f-name*	*age*	*l-name*	*f-name*	*ID*
Bouvier	**Selma**	40	**Bouvier**	**Selma**	1232
Bouvier	**Selma**	40	Smith	**Selma**	4423
Bouvier	Patty	2	**Bouvier**	Selma	1232
Bouvier	Patty	40	Smith	Selma	4423
Smith	Maggie	2	Bouvier	Selma	1232
Smith	Maggie	2	**Smith**	Selma	4423

We remove duplicate attributes…

l-name	*f-name*	*age*	*l-name*	*f-name*	*ID*
Bouvier	Selma	40	Bouvier	Selma	1232

The natural join of *A* and *B*

Note that this is just a way to visualize the natural join, we don't really have to do the cross product as in this example

$A \bowtie B =$

l-name	*f-name*	*age*	*ID*
Bouvier	Selma	40	1232

Natural-Join Operation

- Notation: $r \bowtie s$
- Let r and s be relation instances on schemas R and S respectively.The result is a relation on schema $R \cup S$ which is obtained by considering each pair of tuples t_r from r and t_s from s.
- If t_r and t_s have the same value on each of the attributes in $R \cap S$, a tuple t is added to the result, where
  - t has the same value as t_r on r
  - t has the same value as t_s on s
- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$
- Result schema = (A, B, C, D, E)
- $r \bowtie s$ is defined as:

$$\Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}\ (\sigma_{r.B = s.B\ \ r.D = s.D}\ (r \times s))$$

Natural Join Operation – Example

- Relation instances *r*, *s*:

A	*B*	*C*	*D*
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	*D*	*E*
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

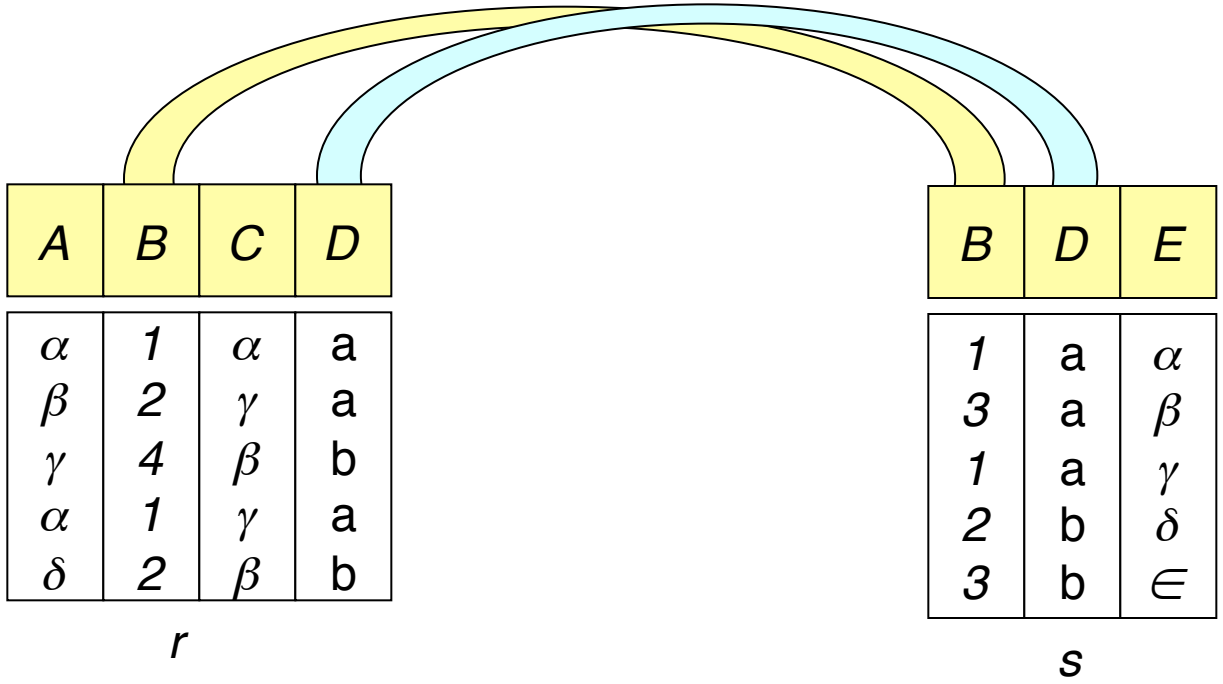
s

$r \bowtie s$

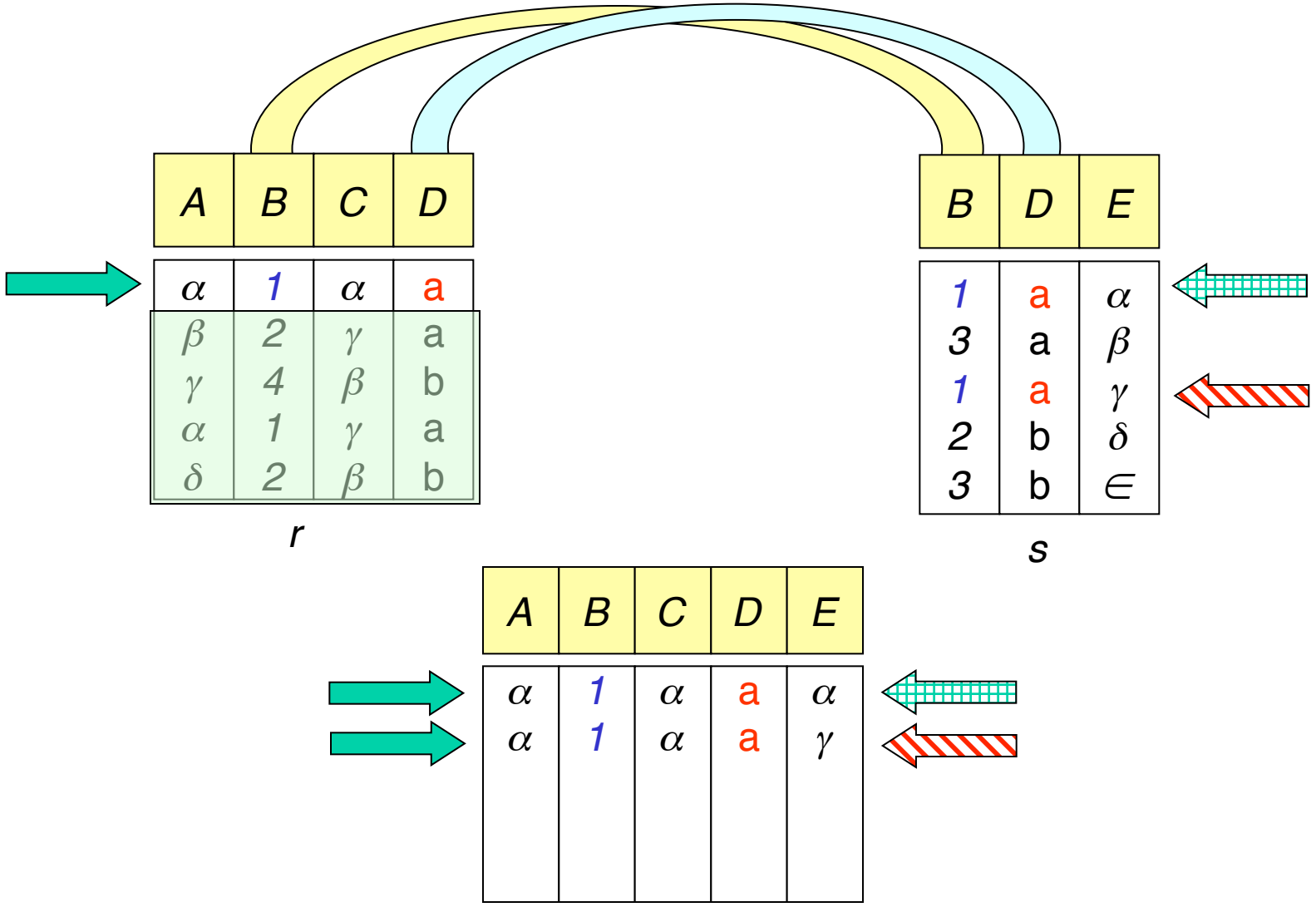
A	*B*	*C*	*D*	*E*
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

How did we get here?

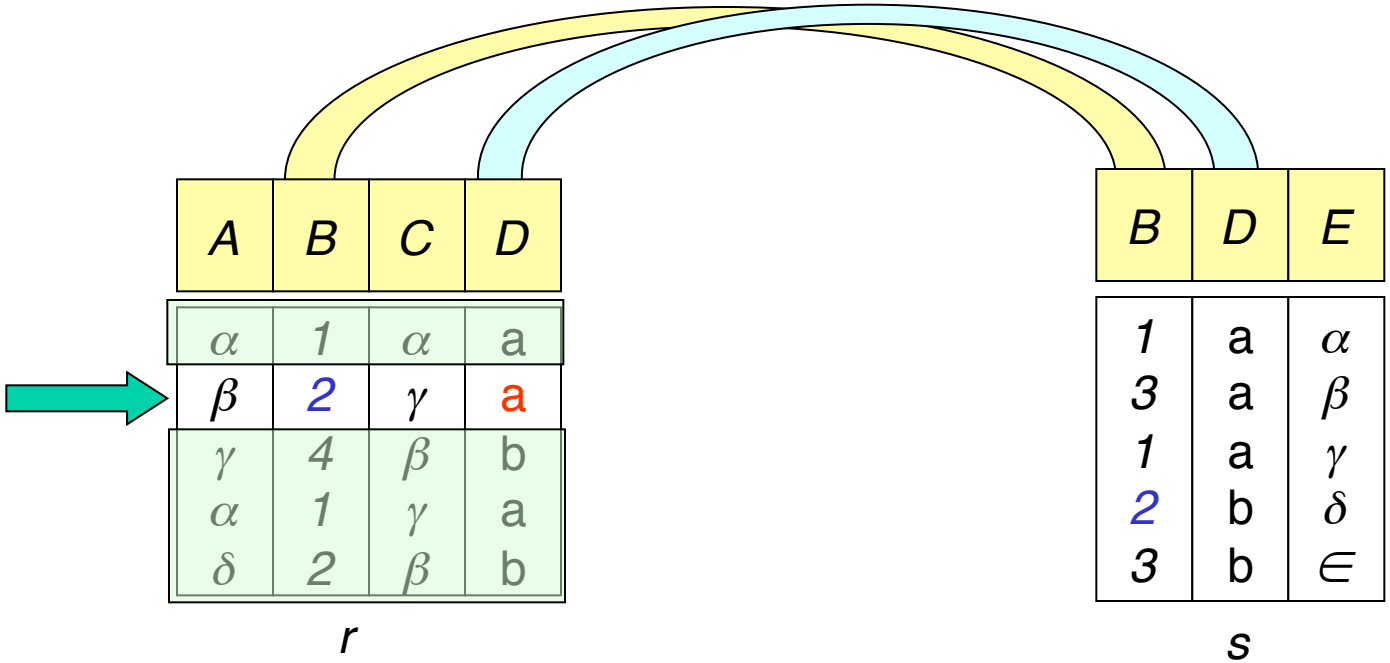
Lets do a trace over the next few slides…



First we note which attributes the two relations have in common…

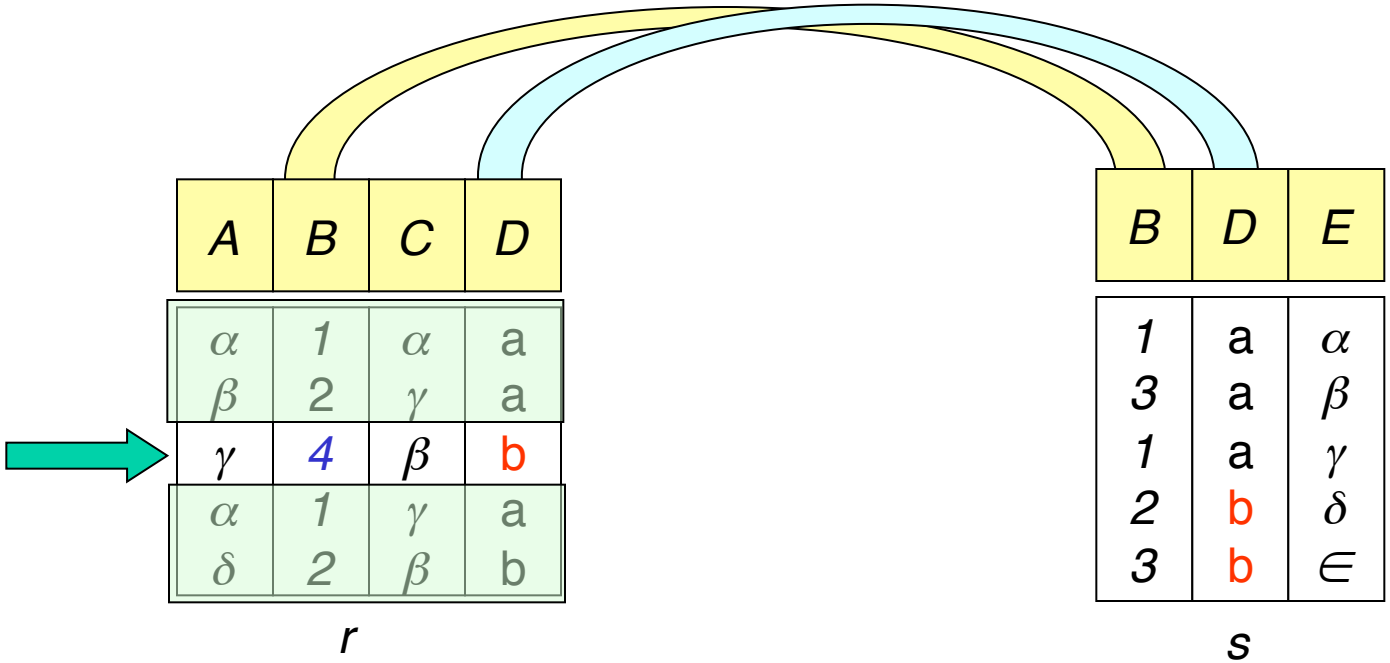


There are two rows in s that match our first row in r, (in the relevant attributes) so both are joined to our first row…



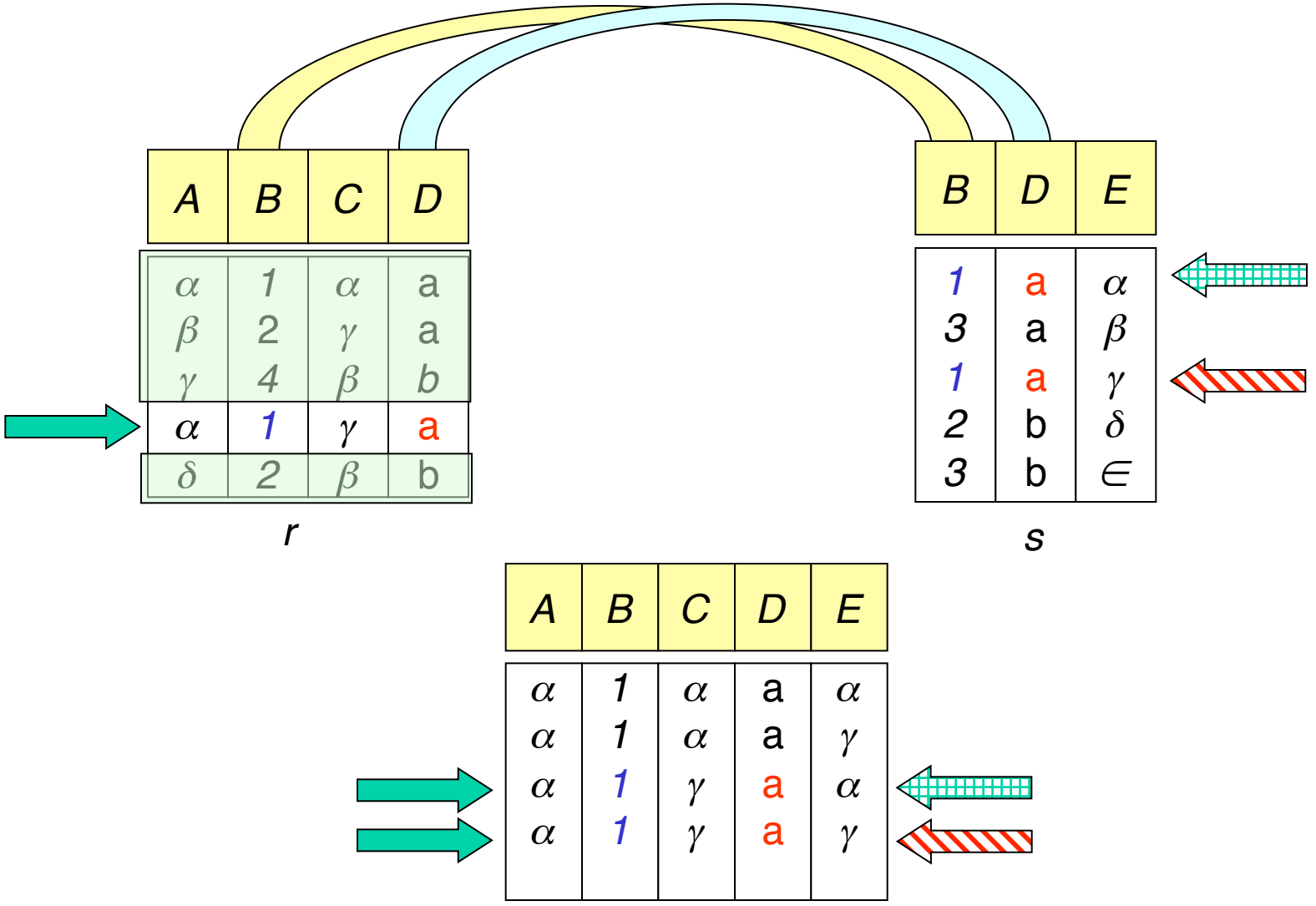
A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ

…there are no rows in s that match our second row in r, so do nothing…

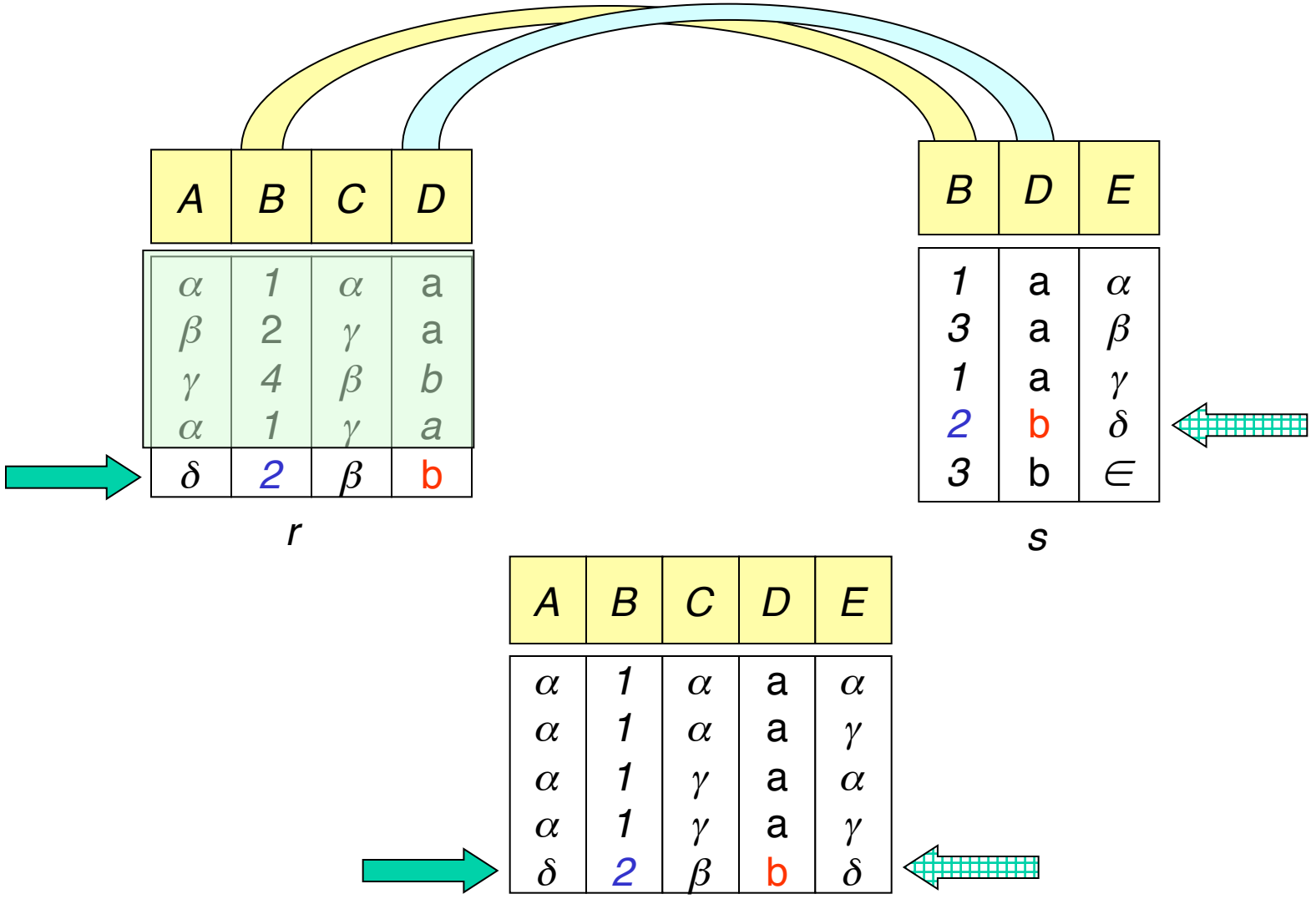


A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ

…there are no rows in s that match our third row in r, so do nothing…



There are two rows in s that match our fourth row in r, so both are joined to our fourth row…



There is one row that matches our fifth row in r,.. so it is joined to our fifth row and we are done!

Natural Join on Sailors Example

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

$S1 \bowtie R1 =$

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

Earlier We Saw…

Query: Find the name of the sailor who reserved boat 101.

$$Temp = \rho(sid \rightarrow sid1, S1) \times \rho(sid \rightarrow sid2, R1)$$

$$\text{Result} = \pi_{Sname}(\sigma_{sid1 = sid2 \wedge bid = 101}(Temp))$$

* Note my use of "temporary" relation Temp.

Query revisited using natural join

Query: Find the name of the sailor who reserved boat 101.

$$\text{Result} = \pi_{Sname}(\sigma_{bid=101}(S1 \bowtie R1))$$

Or

$$\text{Result} = \pi_{Sname}(S1 \bowtie \sigma_{bid=101}(R1))$$

What's the difference between these two approaches?

Conditional-Join Operation:

The conditional join is actually the most general type of join. I introduced the natural join first only because it is more intuitive and... natural!

Just like natural join, conditional join combines a cross product and a selection into one operation. However instead of only selecting rows that have equality on those attributes that appear in both relation schemes, we allow selection based on any predicate.

$$r \bowtie_c s = \sigma_c(r \times s)$$

Where c is any predicate the attributes of r and/or s

Duplicate rows are removed as always, but duplicate columns are not removed!

Conditional-Join Example:

We want to find all women that are younger than their husbands…

r

l-name	*f-name*	*marr-Lic*	*age*
Simpson	Marge	777	35
Lovejoy	Helen	234	38
Flanders	Maude	555	24
Krabappel	Edna	978	40

s

l-name	*f-name*	*marr-Lic*	*age*
Simpson	Homer	777	36
Lovejoy	Timothy	234	36
Simpson	Bart	*null*	9

$r \bowtie_{r.age < s.age \text{ AND } r.Marr\text{-}Lic = s.Marr\text{-}Lic} s$

r.l-name	*r.f-name*	*r.Marr-Lic*	*r.age*	*s.l-name*	*s.f-name*	*s.marr-Lic*	*s.age*
Simpson	Marge	777	35	Simpson	Homer	777	36

Note we have removed ambiguity of attribute names by using "dot" notation
Also note the redundant information in the *marr-lic* attributes

Equi-Join

- Equi-Join: Special case of conditional join where the conditions consist only of equalities.
- Natural Join: Special case of equi-join in which equalities are specified on ALL fields having the same names in both relations.

Equi-Join

r

l-name	*f-name*	*marr-Lic*	*age*
Simpson	Marge	777	35
Lovejoy	Helen	234	38
Flanders	Maude	555	24
Krabappel	Edna	978	40

s

l-name	*f-name*	*marr-Lic*	*age*
Simpson	Homer	777	36
Lovejoy	Timothy	234	36
Simpson	Bart	*null*	9

$r \bowtie_{r.Marr\text{-}Lic = s.Marr\text{-}Lic} s$

r.l-name	*r.f-name*	*Marr-Lic*	*r.age*	*s.l-name*	*s.f-name*	*s.age*
Simpson	Marge	777	35	Simpson	Homer	36
Lovejoy	Helen	234	38	Lovejoy	Timothy	36

Review on Joins

- All joins combine a cross product and a selection into one operation.
- Conditional Join
  - the selection condition can be of any predicate (e.g. rating1 > rating2)
- Equi-Join:
  - Special case of conditional join where the conditions consist only of equalities.
- Natural Join
  - Special case of equi-join in which equalities are specified on ALL fields having the same names in both relations.