RELATIONAL ALGEBRA

Relational Algebra

- Procedural language
- Five basic operators
    - **selection**
    - **projection**
    - **union**
    - **set difference**
    - **Cross 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.

Selection 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 **selection** 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.

Selection Operation

- Notation: $\sigma_{p}(r)$

lowercase Greek sigma σ

- *p* is called the **selection** predicate
- Defined as:

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

Where *p* is a formula in propositional calculus consisting of terms connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each term is one of:

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

where *op* is one of: $=, \neq, >, \geq, <, \leq$

- Example of selection:

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

Projection 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 capital letter pi

A	*C*
α	*7*
α	*1*
β	*1*
β	*2*

Intuition: The **projection** 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*.

Projection 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 projection operation removes duplicate rows, since relations are sets.

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

Projection Operation

- Notation:

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

Greek capital letter 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 $A = \alpha$ and $B = 2$. So one was discarded.

Union Operation

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

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

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)

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:

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

- Set differences must be taken between *compatible* relations.
  - *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 $r \times s$

- Defined as:

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

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \emptyset$).
- 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 C"

$\sigma_{A=C}(r \times s) \Rightarrow$

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

Rename Operation

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

Example:

$$\rho_{myRelation}(r - s)$$

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 *E* has arity *n*, then

$$\rho_{X\,(A1,\, A2,\, \ldots,\, An)}\,(E)$$

returns the result of expression *E* under the name *X*, and with the attributes renamed to *A1, A2, …., An*.

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

Example

$$\rho_{myRelation(E,K)}\,(r - s)$$

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

E	K
α	1
β	1

myRelation

Banking Examples

branch (branch-name, branch-city, assets)

customer (customer-name, customer-street, customer-only)

account (account-number, branch-name, balance)

loan (loan-number, branch-name, amount)

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

Note that I have not indicated primary keys here for simplicity.

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

Example Queries

- Find all loans of over $1200

$\sigma_{amount > 1200}(loan)$

"select from the relation *loan*, only the rows which have a *amount* greater than 1200"

loan

loan-number	***branch-name***	***amount***
1234	Riverside	1,923.03
3421	Irvine	123.00
2342	Dublin	56.25
4531	Prague	120.03

$\sigma_{amount > 1200}(loan)$

1234	Riverside	1,923.03

Example Queries

- Find the loan number for each loan of an amount greater than $1200

$$\Pi_{loan\text{-}number}\ (\sigma_{amount\ >\ 1200}\ (loan))$$

"select from the relation *loan*, only the rows which have a *amount* greater than 1200, then project out just the *loan_number*"

loan

loan-number	*branch-name*	*amount*
1234	Riverside	1,923.03
3421	Irvine	123.00
2342	Dublin	56.25
4531	Prague	120.03

$\sigma_{amount} > 1200\ (loan)$

1234	Riverside	1,923.03

$\Pi_{loan\text{-}number}\ (\sigma_{amount} > 1200\ (loan))$

1234

Example Queries

- Find all loans greater than \$1200 or less than \$75

$$\sigma_{amount\ >\ 1000\ \textbf{or}\ amount\ <\ 75}(loan)$$

“select from the relation *loan*, only the rows which have a *amount* greater than 1000 or an *amount* less than 75

loan

loan-number	*branch-name*	*amount*
1234	Riverside	1,923.03
3421	Irvine	123.00
2342	Dublin	56.25
4531	Prague	120.03

$$\sigma_{amount\ >\ 1000\ \textbf{or}\ amount\ <\ 75}(loan)$$

1234	Riverside	1,923.03
2342	Dublin	56.25

Example Queries

- Find the names of all customers who have a loan, an account, or both, from the bank

$\Pi_{customer\text{-}name}$ *(borrower)* $\cup$ $\Pi_{customer\text{-}name}$ *(depositor)*

borrower

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

depositor

customer-name	***account-number***
Moe	3467
Apu	2312
Patty	9999
Krusty	3423

$\Pi_{customer\text{-}name}$ *(borrower)*

Patty
Apu
Selma
Ned

Moe
Apu
Patty
Krusty
Selma
Ned

$\Pi_{customer\text{-}name}$ *(depositor)*

Moe
Apu
Patty
Krusty

Example Queries

Find the names of all customers who have a loan at the Riverside branch.

$\Pi_{\textit{customer-name}} (\sigma_{\textit{branch-name=“Riverside ”}} (\sigma_{\textit{borrower.loan-number = loan.loan-number}}(\textit{borrower x loan})))$

Note this example is split over two slides!

We retrieve *borrower* and *loan*…

borrower

customer-name	***loan-number***
Patty	1234
Apu	3421

loan

loan-number	***branch-name***	***amount***
1234	Riverside	1,923.03
3421	Irvine	123.00

…we calculate their cross product…

customer-name	***borrower.loan-number***	***loan.loan-number***	***branch-name***	***amount***
Patty	1234	1234	Riverside	1,923.03
Patty	1234	3421	Irvine	123.00
Apu	3421	1234	Riverside	1,923.03
Apu	3421	3421	Irvine	123.00

$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=\text{“}Riverside\text{”}}\ (\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower\ x\ loan)))$

…we calculate their cross product…

customer-name	***borrower.loan-number***	***loan.loan-number***	***branch-name***	***amount***
Patty	1234	1234	Riverside	1,923.03
Patty	1234	3421	Irvine	123.00
Apu	3421	1234	Riverside	1,923.03
Apu	3421	3421	Irvine	123.00

…we select the rows where *borrower.loan-number* is equal to *loan.loan-number*…

customer-name	***borrower.loan-number***	***loan.loan-number***	***branch-name***	***amount***
Patty	1234	1234	Riverside	1,923.03
Apu	3421	3421	Irvine	123.00

…we select the rows where *branch-name* is equal to *“Riverside”*

customer-name	***borrower.loan-number***	***loan.loan-number***	***branch-name***	***amount***
Patty	1234	1234	Riverside	1,923.03

…we project out the *customer-name.*

Patty

Example Queries

Note this example is split over three slides!

Find the largest account balance

...we will need to rename *account* relation as *d*...

$\Pi_{balance}(account) - \Pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho_d(account)))$

We do a rename to get a "copy" of *account* which we call *d*…

… next we will do a cross product…

account

account-number	***balance***
Apu	100.30
Patty	12.34
Lenny	45.34

d

account-number	***balance***
Apu	100.30
Patty	12.34
Lenny	45.34

$\Pi_{balance}(account) - \Pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho_d(account)))$

… do a cross product…

account.account-number	*account. balance*	*d.account -number*	*d.balance*
Apu	100.30	Apu	100.30
Apu	100.30	Patty	12.34
Apu	100.30	Lenny	45.34
Patty	12.34	Apu	100.30
Patty	12.34	Patty	12.34
Patty	12.34	Lenny	45.34
Lenny	45.34	Apu	100.30
Lenny	45.34	Patty	12.34
Lenny	45.34	Lenny	45.34

…select out all rows where *account.balance* is less than *d.balance*…

account.account-number	*account. balance*	*d.account -number*	*d.balance*
Patty	12.34	Apu	100.30
Patty	12.34	Lenny	45.34
Lenny	45.34	Apu	100.30

.. next we project…

$\Pi_{balance}(account) - \Pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho_d(account)))$

account.account-number	*account.balance*	*d.account-number*	*d.balance*
Patty	12.34	Apu	100.30
Patty	12.34	Lenny	45.34
Lenny	45.34	Apu	100.30

.. next we project out *account.balance*…

account.balance
12.34
45.34

…then we do a set difference between it and the original *account.balance* from the account relation…

account

account-number	*balance*
Apu	100.30
Patty	12.34
Lenny	45.34

… the set difference leaves us with one number, the largest value!

100.30

Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\Pi_S(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_x (E_1)$, x is the new name for the result of E_1

Important note on notation!

When you see operators between lowercase letters…

$r - s$

They refer to relation algebra operators

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

A	B
α	1
β	1

r-s

When you see operators between uppercase letters…

$R - S$

They refer to operators on schemas

$$R = (age, name)$$
$$S = (name)$$

$$R\text{-}S \quad = \quad (age)$$

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
- Division
- Set intersection

All joins are really special cases of conditional join

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 is 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…

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

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*

$A \bowtie B =$

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

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

Natural-Join Operation

- Notation: $r \bowtie s$
- Let r and s be relations 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

- Relations *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	$\in$

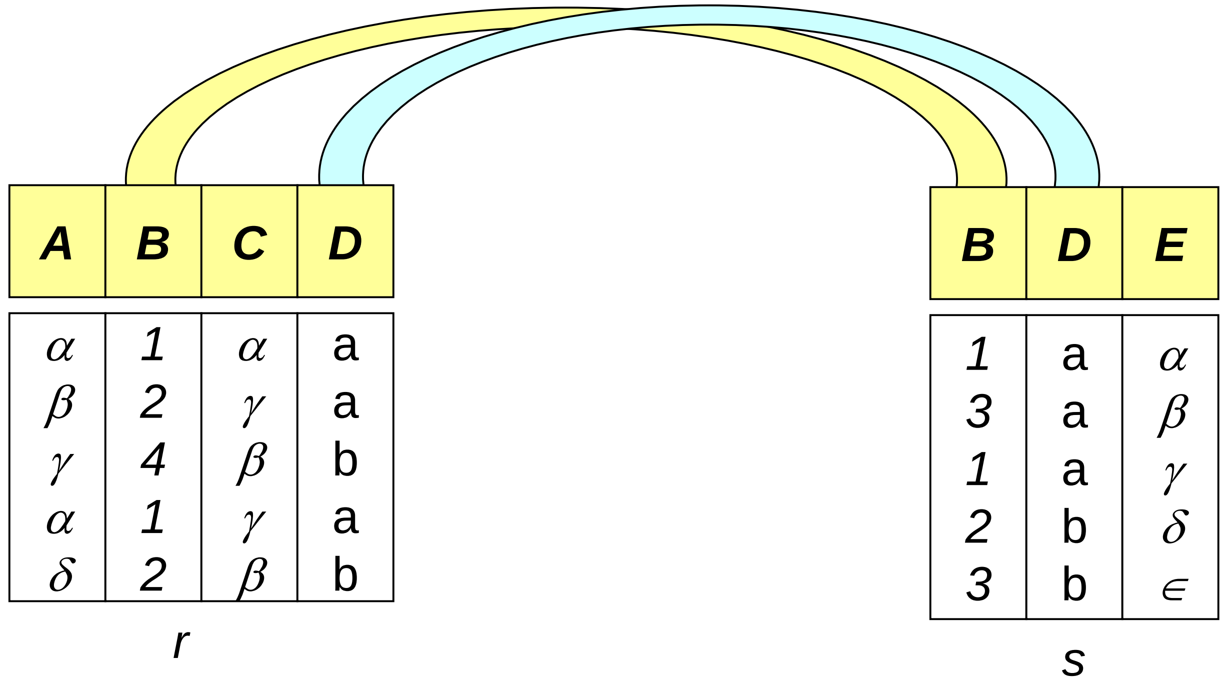
s

$r \bowtie s$

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…



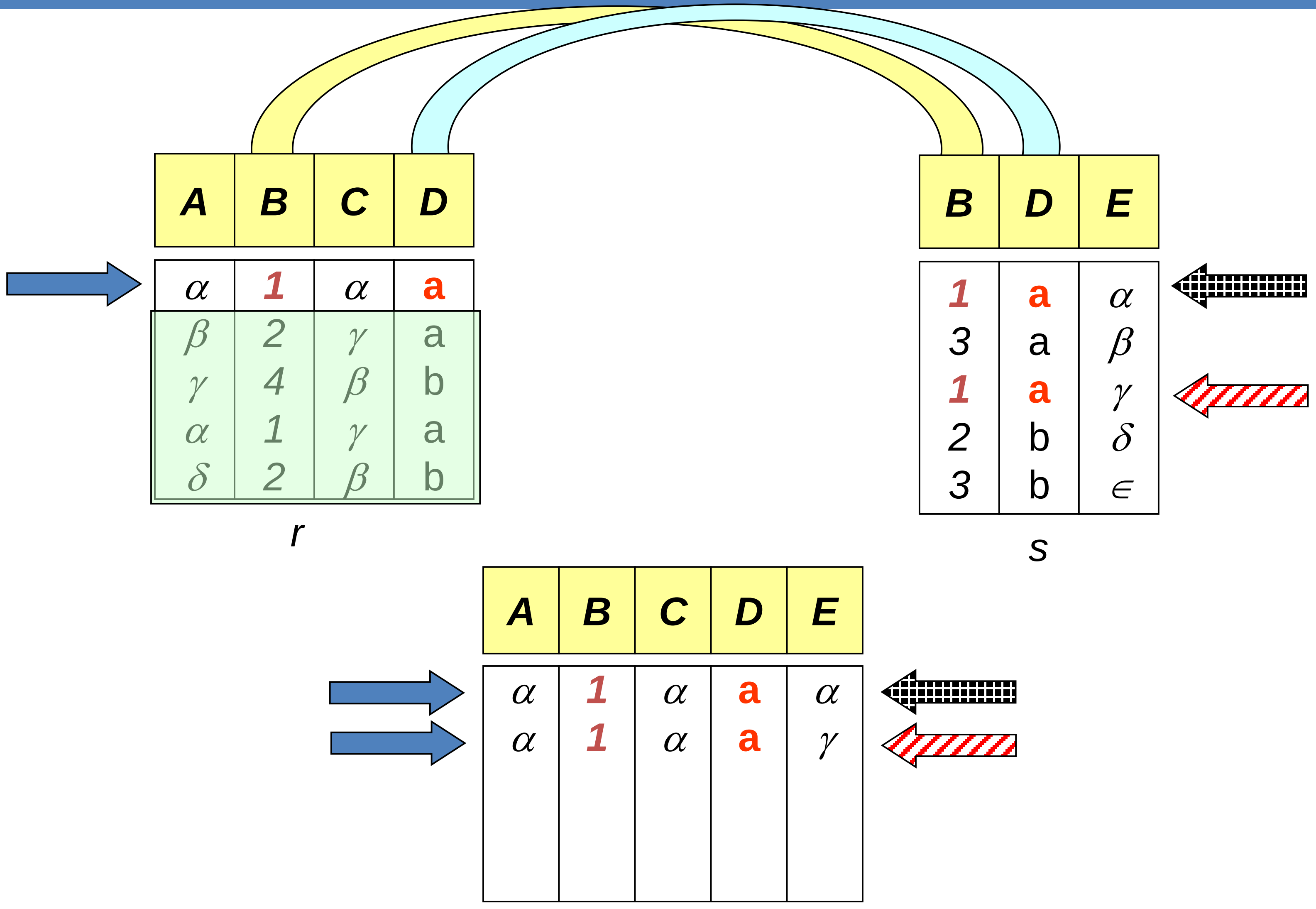
A	*B*	*C*	*D*
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

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

s

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



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

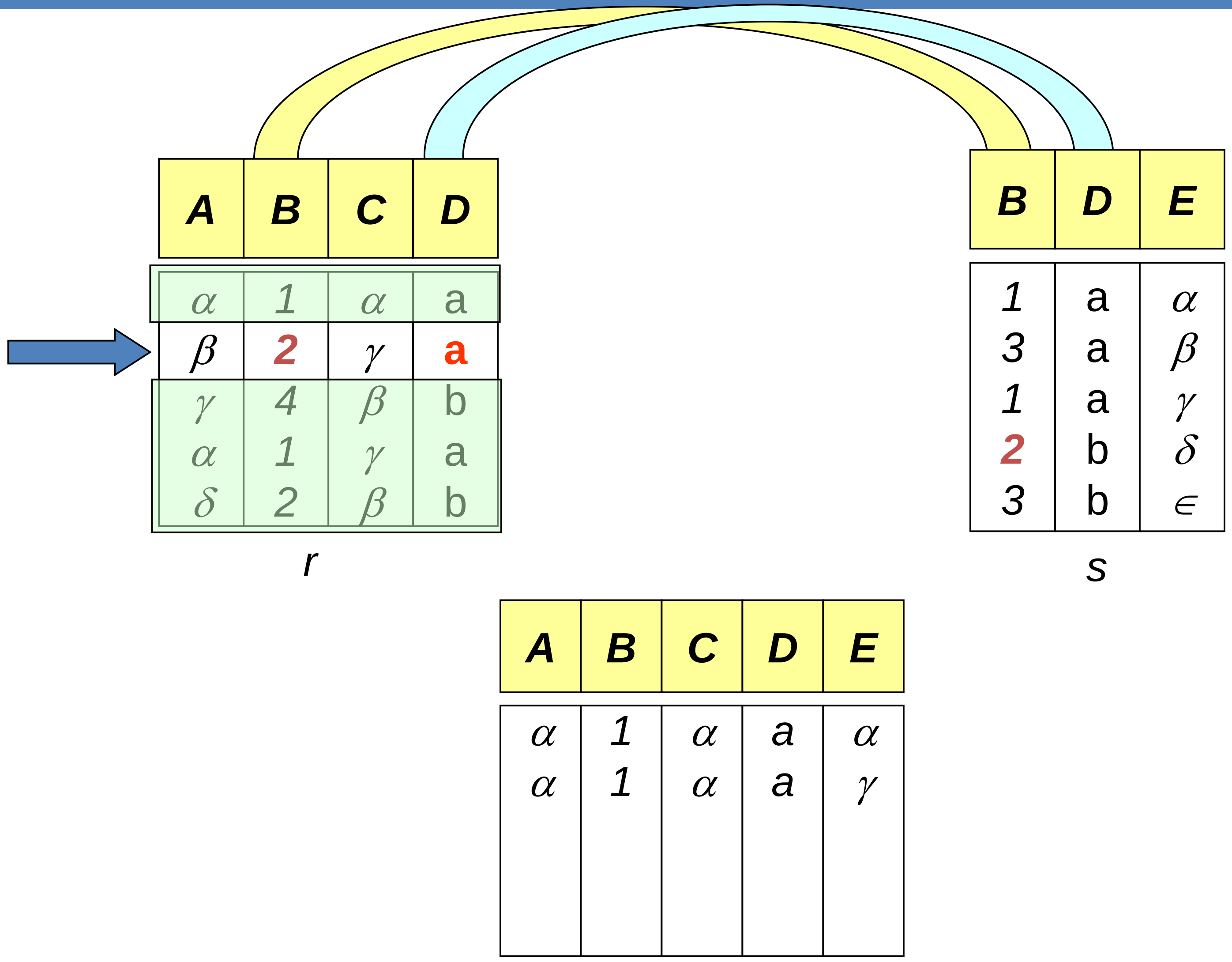
r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	∈

s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ

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
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

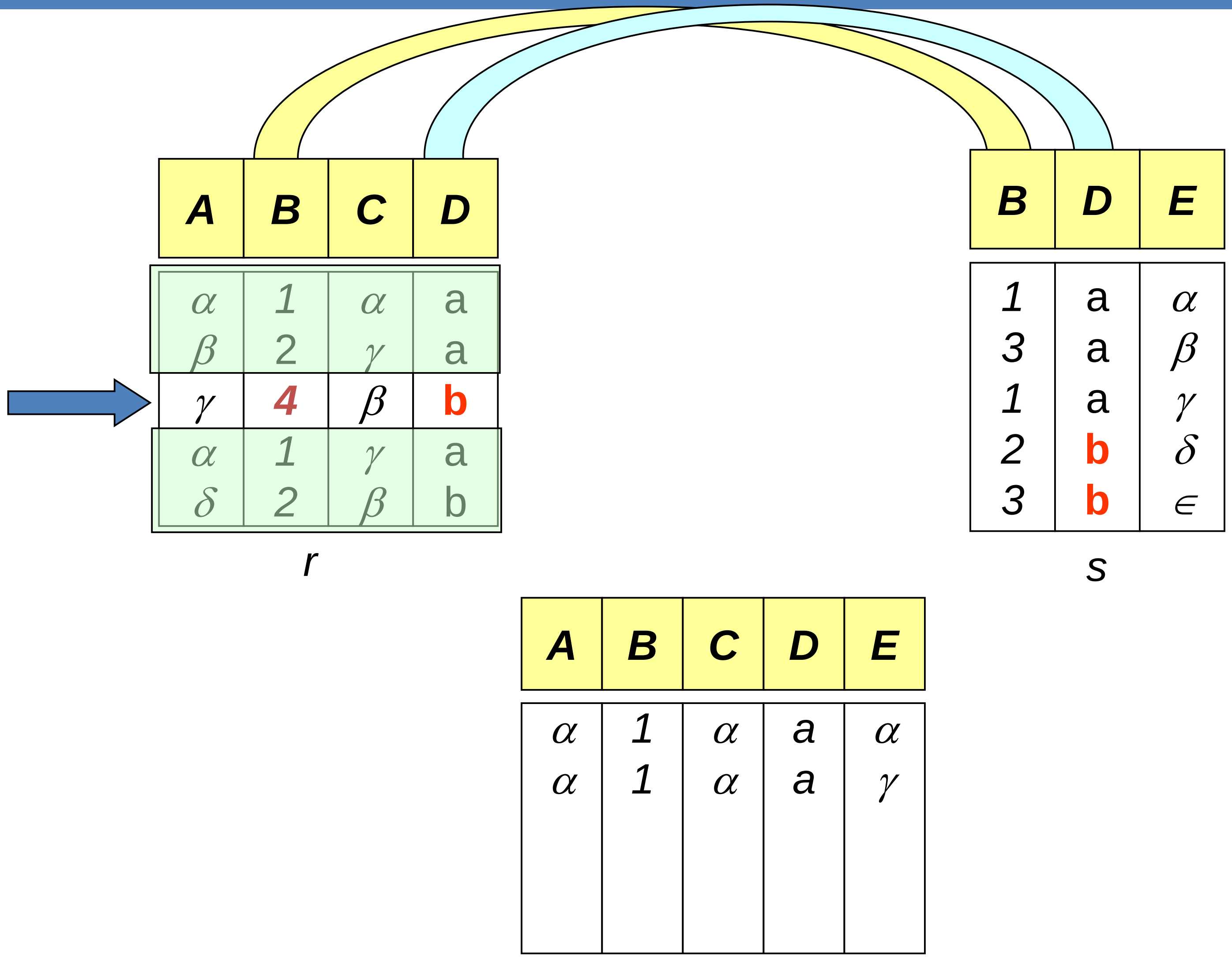
r

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

s

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
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

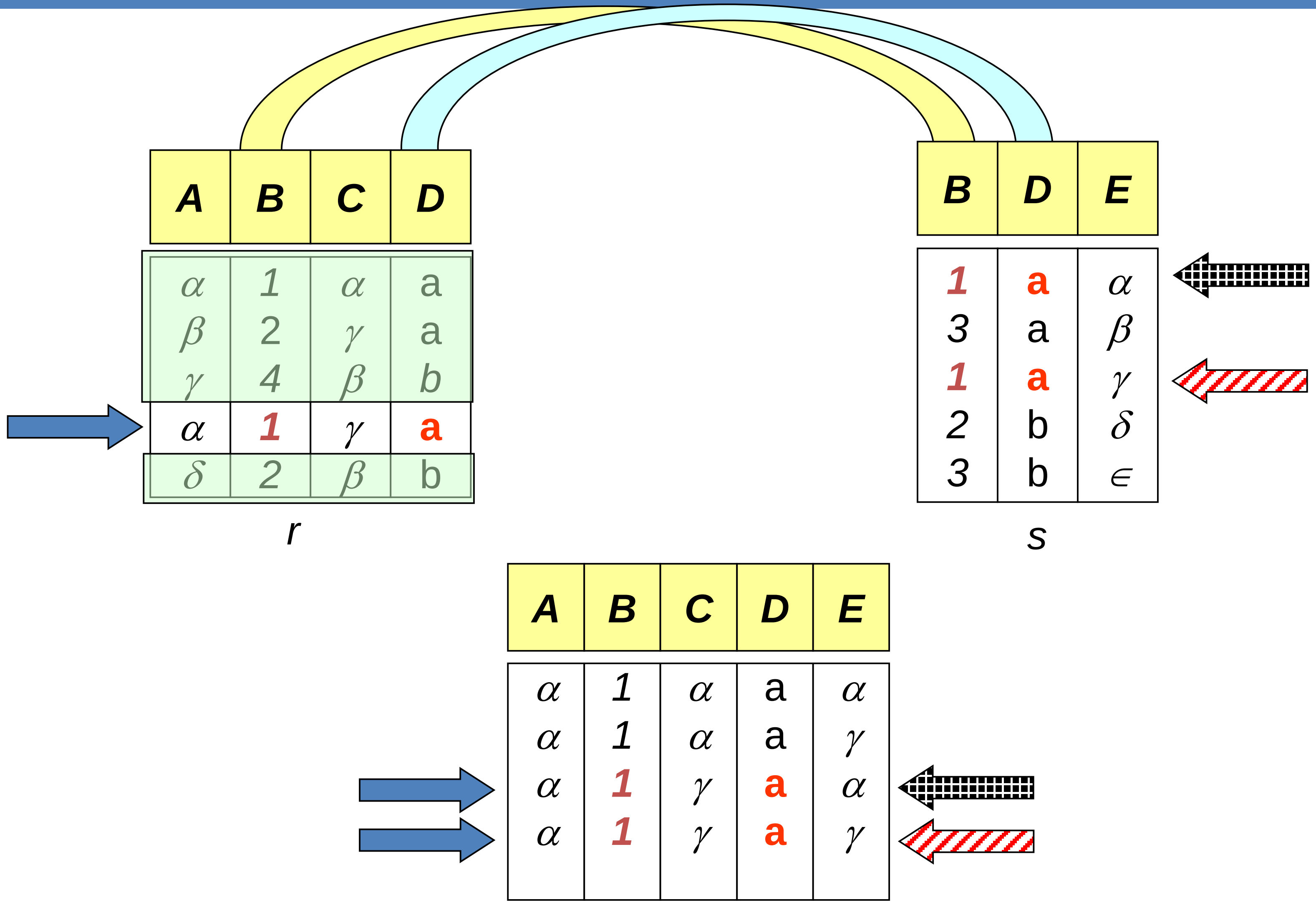
r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	∈

s

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…



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	∈

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

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

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	∈

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

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

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 older 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

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

r.l-name	*r.f-name*	***r.Marr-Lic***	*r.age*	*s.l-name*	*s.f-name*	***s.marr-Lic***	*s.age*
Lovejoy	Helen	234	38	Lovejoy	Timothy	234	36

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

Set-Intersection Operation - Example

Relation *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

$r \cap s$

A	B
α	2

Intuition: The **intersection** operation returns all the rows that are in both *r* and *s*.

Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$
- Assume:
  - r, s have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$

Division Operation

$r \ / \ s$

- Suited to queries that include the phrase "for all".
- Let *r* and *s* be relations on schemas R and S respectively where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

  The result of r / s is a relation on schema

  $R - S = (A_1, \ldots, A_m)$

$$r \ / \ s = \{ \ t \ | \ t \in \Pi_{R-S}(r) \land \forall \ u \in s \ (\ tu \in r \ ) \ \}$$

Division Operation – Example

Relations *r*, *s*:

A	***B***
α	*1*
α	*2*
α	3
β	*1*
γ	*1*
δ	*1*
δ	3
δ	4
$\in$	6
$\in$	*1*
β	*2*

r

B
1
2

s

r / *s*:

A
α
β

α occurs in the presence of both ***1*** and ***2***, so it is returned.
β occurs in the presence of both ***1*** and ***2***, so it is returned.
γ does not occur in the presence of both ***1*** and ***2***, so is ignored.
...

Another Division Example

Relations *r*, *s*:

A	*B*	*C*	*D*	*E*
α	a	α	a	*1*
α	a	γ	**a**	**1**
α	a	γ	**b**	**1**
β	a	γ	a	*1*
β	a	γ	b	*3*
γ	a	γ	**a**	**1**
γ	a	γ	**b**	**1**
γ	a	β	b	*1*

r

D	*E*
a	**1**
b	**1**

s

r /*s*:

A	*B*	*C*
α	a	γ
γ	a	γ

<α, a , γ > occurs in the presence of both <***a,1***> and <***b,1***>, so it is returned.
< γ, a , γ > occurs in the presence of both <***a,1***> and <***b,1***>, so it is returned.
<β, a , γ > does not occur in the presence of both <***a,1***> and <***b,1***>, so it is ignored.

Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries, write query as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as a result of the query.
- Assignment must always be made to a temporary relation variable.
- Example: Write *r* / *s* as

$$temp1 \leftarrow \Pi_{R-S}(r)$$
$$temp2 \leftarrow \Pi_{R-S}((temp1 \text{ x } s) - \Pi_{R-S,S}(r))$$
$$result = temp1 - temp2$$

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.
  - May use variable in subsequent expressions.

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are (roughly speaking) **false** by definition.
    - Will study precise meaning of comparisons with nulls later

Outer Join – Example

- Relation *loan*

loan-number	***branch-name***	***amount***
L-170	Springfield	3000
L-230	Shelbyville	4000
L-260	Dublin	1700

- Relation *borrower*

customer-name	***loan-number***
Simpson	**L-170**
Wiggum	**L-230**
Flanders	L-155

Outer Join – Example

- **Inner Join**

  loan *Borrower*

  $\bowtie$

loan-number	***branch-name***	***amount***	***customer-name***
L-170	Springfield	3000	Simpson
L-230	Shelbyville	4000	Wiggum

loan-number	*branch-name*	*amount*
L-170	Springfield	3000
L-230	Shelbyville	4000
L-260	Dublin	1700

customer-name	*loan-number*
Simpson	**L-170**
Wiggum	**L-230**
Flanders	L-155

- **Left Outer Join**

  loan $⟕$ *borrower*

loan-number	***branch-name***	***amount***	***customer-name***
L-170	Springfield	3000	Simpson
L-230	Shelbyville	4000	Wiggum
L-260	Dublin	1700	*null*

Outer Join – Example

Right Outer Join

loan *borrower*

⟖

loan-number	***branch-name***	***amount***	***customer-name***
L-170	Springfield	3000	Simpson
L-230	Shelbyville	4000	Wiggum
L-155	*null*	*null*	Flanders

loan-number	*branch-name*	*amount*
L-170	Springfield	3000
L-230	Shelbyville	4000
L-260	Dublin	1700

customer-name	*loan-number*
Simpson	**L-170**
Wiggum	**L-230**
Flanders	L-155

Full Outer Join

loan ⟗ *borrower*

loan-number	***branch-name***	***amount***	***customer-name***
L-170	Springfield	3000	Simpson
L-230	Shelbyville	4000	Wiggum
L-260	Dublin	1700	*null*
L-155	*null*	*null*	Flanders

Relational Calculus in 3 slides!

- *Query* has the form:

$$\left\{ \langle x1, x2, \ldots, xn \rangle \mid p\left(\langle x1, x2, \ldots, xn \rangle \right) \right\}$$

Users define queries in terms of what they want, not in terms of how to compute it.

- *Answer* includes all tuples $\langle x1, x2, \ldots, xn \rangle$ that make the *formula* $p\left(\langle x1, x2, \ldots, xn \rangle \right)$ be *true*.

- *Formula* is recursively defined, starting with simple *atomic formulas* (getting tuples from relations or making comparisons of values), and building bigger and better formulas using the *logical connectives*.

Domain Relational Calculus Formulas

- *Atomic formula:*
  - $\langle x1, x2, ..., xn \rangle \in Rname$ or X *op* Y, or X *op* constant
  - *op* is one of [illegible]
- *Formula:*
  - an atomic formula, or
  - [illegible], where p and q are formulas, or
  - [illegible] where variable X is *free* in p(X), or
  - [illegible] where variable X is *free* in p(X)
- The use of quantifiers [illegible] and [illegible] is said to <u>*bind*</u> X.
  - A variable that is not bound is free.

$\exists X \quad \forall X$

– The variables x1, ..., xn that appear to the left of `|' must be the *only* free variables in the formula p(...).

Example

Find the names of all customers who have a loan at the Riverside branch.

$\Pi_{customer\text{-}name}$ $(\sigma_{branch\text{-}name=\text{“Riverside ”}}$ $(\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower \times loan)))$

borrower

customer-name	***loan-number***
Patty	1234
Apu	3421

loan

loan-number	***branch-name***	***amount***
1234	Riverside	1,923.03
3421	Irvine	123.00

$\{\langle X\rangle \mid \langle X,Y\rangle \in \text{borrower} \wedge \exists A,B,C(\langle A,B,C\rangle \in \text{loan } B = \text{‘Riverside’} \wedge Y=A)\}$