Relational Query Languages: Relational Algebra

Juliana Freire

Relational Query Languages

- *Query languages:* Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
  - Simple data structure – sets!
    - Easy to understand, easy to manipulate
  - 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.

Describing a Relational Database Mathematically: Relational Algebra

- We can describe tables in a relational database as sets of tuples
- We can describe query operators using set theory
- The query language is called relational algebra
- Normally, not used directly -- foundation for SQL and query processing
  - SQL adds syntactic sugar
- Very useful for representing execution plans

What is an "Algebra"

- Mathematical system consisting of:
  - *Operands* --- variables or values from which new values can be constructed
  - *Operators* --- symbols denoting procedures that construct new values from given values
- Expressions can be constructed by applying operators to atomic operands and/or other expressions
  - Operations can be **composed** -- algebra is closed
  - Parentheses are needed to group operators

Basics of Relational Algebra

- Algebra of arithmetic: operands are variables and constants, and operators are the usual arithmetic operators
  - E.g., (x+y)*2 or ((x+7)/(y-3)) + x
- Relational algebra: operands are variables that stand for relations and relations (sets of tuples), and operators are designed to do the most common things we need to do with relations in databases, e.g., *union, intersection, selection, projection, Cartesian product, etc*
  - E.g., ($\pi_{\text{c-owner}}$Checking-account) $\cap$ ($\pi_{\text{s-owner}}$Savings-account)
- The result is an algebra that can be used as a *query language* for relations.

Basics of Relational Algebra (cont.)

- A query is applied to *relation instances*, and the *result of a query is also a relation instance*
- Algebra is "closed" → operations can be *composed*
  - *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.
- Operators refer to relation attributes by position or name:
  - E.g., Account(number, owner, balance, type)

  Positional←Account.$1 = Account.number → Named field

  Positional←Account.$3 = Account.balance → Named field
  - Positional notation easier for formal definitions, named-field notation more readable.

Relational Algebra: Operations

- The usual set operations: union, intersection, difference
  - Both operands must have the same relation schema
- Operations that remove parts of relations:
  - Selection: pick certain rows
  - Projection: pick certain columns
- Operations that combine tuples from two relations: Cartesian product, join
- Renaming of relations and attributes

Removing Parts of Relations

σ (sigma)

Selection: Example

σ_c R= select -- produces a new relation with the subset of the tuples in R that match the condition *C*

Sample query: $\sigma_{\text{Type = "savings"}}$ Account

Account	Number	Owner	Balance	Type
	~~101~~	~~J. Smith~~	~~1000.00~~	~~checking~~
	~~102~~	~~W. Wei~~	~~2000.00~~	~~checking~~
	103	J. Smith	5000.00	savings
	~~104~~	~~M. Jones~~	~~1000.00~~	~~checking~~
	~~105~~	~~H. Martin~~	~~10,000.00~~	~~checking~~

	Number	Owner	Balance	Type
	103	J. Smith	5000.00	savings

Selection: Another Example

$\sigma_{\text{Balance} < 4000}$ Account

Selects rows that satisfy *selection condition*

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	~~103~~	~~J. Smith~~	~~5000.00~~	~~savings~~
	104	M. Jones	1000.00	checking
	~~105~~	~~H. Martin~~	~~10,000.00~~	~~checking~~

	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	104	M. Jones	1000.00	checki

Schema of result identical to schema of input relation

Projection: Example

π (pi)

$\pi_{\text{AttributeList}}$ R = project -- deletes attributes that are not in *projection list*.

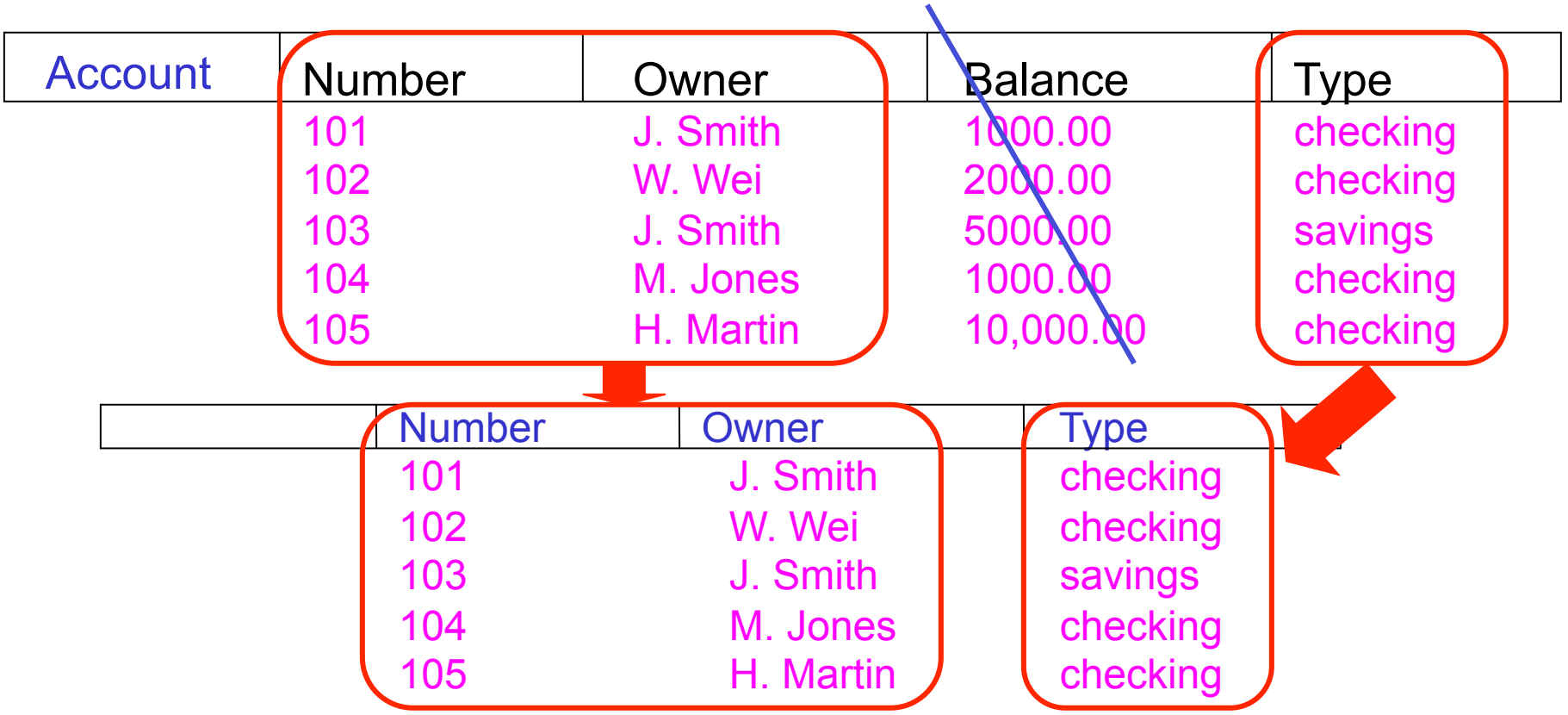
Sample query: $\pi_{\text{Number, Owner, Type}}$ Account

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Projection: Example

π = project

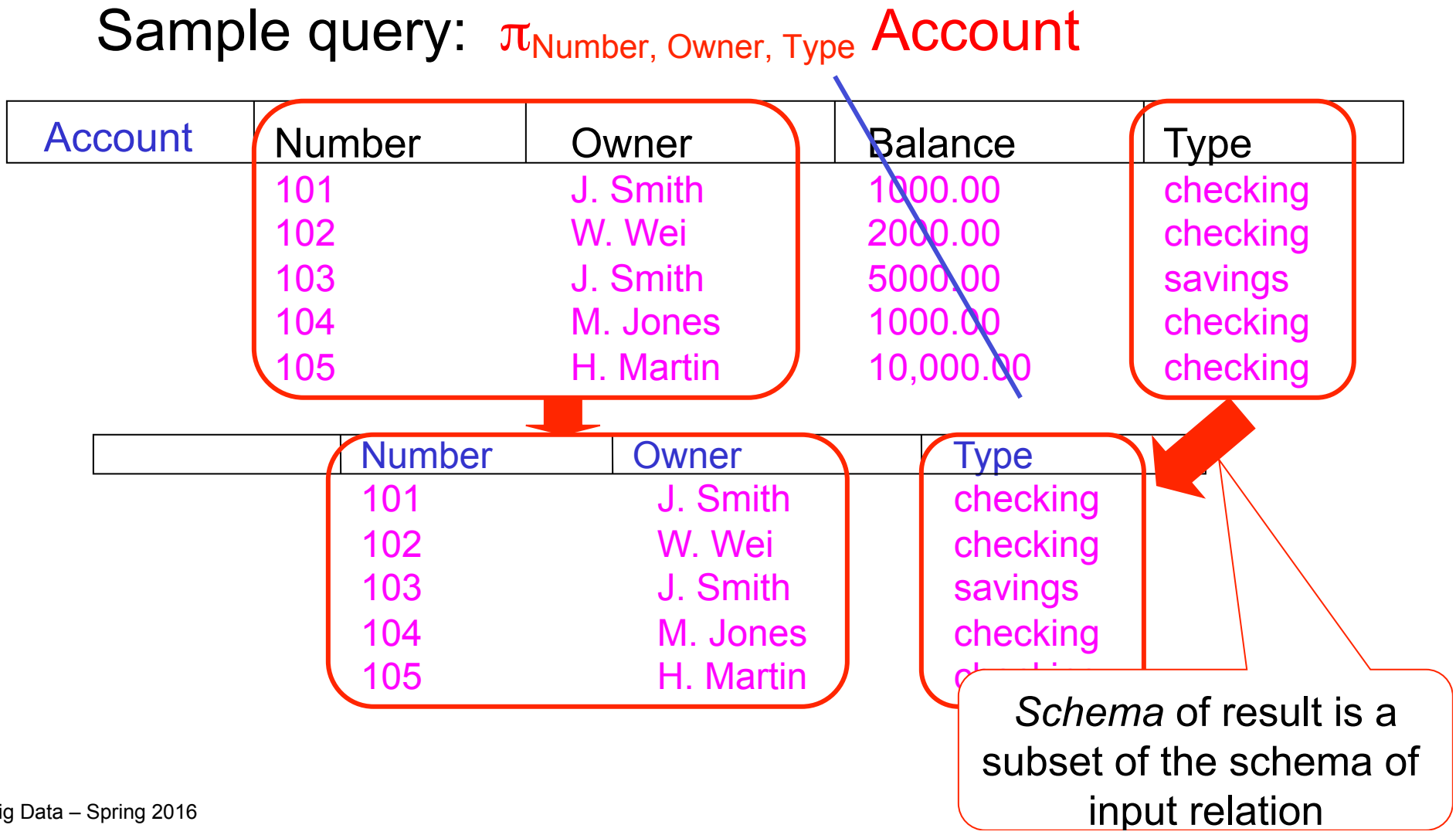
Sample query: $\pi_{\text{Number, Owner, Type}}$ Account



Projection: Example

π = project

Sample query: $\pi_{\text{Number, Owner, Type}}$ Account



Projection: Another Example

π_{Owner} Account

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

	Owner
	J. Smith
	W. Wei
	M. Jones
	H. Martin

Note: Projection operator eliminates *duplicates, Why???*

In a DBMS products, do you think duplicates should be eliminated for every query? Are they?

Extended (Generalized) Projection

- Allows arithmetic functions and duplicate occurrences of the same attribute to be used in the projection list

$$\Pi_{F1, F2, \ldots, Fn}(E)$$

  - E is any relational-algebra expression
  - $F_1, F_2, \ldots, F_n$ are arithmetic expressions involving constants and attributes in the schema of E.
- Given relation *credit-info(customer-name, limit, credit-balance),* find how much more each person can spend:

$$\Pi_{customer\text{-}name,\ limit - credit\text{-}balance}\ (credit\text{-}info)$$

Can use rename to give a name to the column!

$$\Pi_{customer\text{-}name,\ (limit - credit\text{-}balance) \rightarrow credit\text{-}available}\ (credit\text{-}info)$$

Extended Projection: Another Example

R = (

A	B
1	2
3	4

)

$\pi_{A+B->C,A,A}$ (R) =

C	A1	A2
3	1	1
7	3	3

Set Operations

Union: Example

∪ = union

Checking-account ∪ Savings-account

Checking-account	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00

Savings-account	s-num	s-owner	s-balance
	103	J. Smith	5000.00

	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00
	103	J. Smith	5000.00

Union Compatible

- Two relations are *union-compatible* if they have the same degree (i.e., the same number of attributes) and the corresponding attributes are defined on the same domains.
- Suppose we have these tables:

Checking-Account (c-num:str, c-owner:str, c-balance:real)

Savings-Account (s-num:str, s-owner:str, s-balance:real)

These are union-compatible tables.

- *Union, intersection, & difference require union-compatible tables*

Intersection

∩ = intersection

Checking-account ∩ Savings-account

What's the answer to this query?

Checking-account

c-num	c-owner	c-balance
101	J. Smith	1000.00
102	W. Wei	2000.00
104	M. Jones	1000.00
105	H. Martin	10,000.00

Savings-account

s-num	s-owner	s-balance
103	J. Smith	5000.00

Intersection (cont.)

Checking-account

c-num	c-owner	c-balance
101	J. Smith	1000.00
102	W. Wei	2000.00
104	M. Jones	1000.00
105	H. Martin	10,000.00

Savings-account

s-num	s-owner	s-balance
103	J. Smith	5000.00

Checking-account ∩ Savings-account

What's the answer to this query?

It's empty. There are no tuples that are in both tables.

$(\pi_{\text{c-owner}}\text{Checking-account}) \cap (\pi_{\text{s-owner}}\text{Savings-account})$

What's the answer to this new query?

Intersection (cont.)

Checking-account

c-num	c-owner	c-balance
101	J. Smith	1000.00
102	W. Wei	2000.00
104	M. Jones	1000.00
105	H. Martin	10,000.00

Savings-account

s-num	s-owner	s-balance
103	J. Smith	5000.00

Checking-account ∩ Savings-account

What's the answer to this query?

It's empty. There are no tuples that are in both tables.

$(\pi_{\text{c-owner}}\text{Checking-account}) \cap (\pi_{\text{s-owner}}\text{Savings-account})$

What's the answer to this new query?

	c-owner
	J. Smith

Difference

— = difference

Checking-account

c-num	c-owner	c-balance
101	J. Smith	1000.00
102	W. Wei	2000.00
104	M. Jones	1000.00
105	H. Martin	10,000.00

Savings-account

s-num	s-owner	s-balance
103	J. Smith	5000.00

Find all the customers that own a Checking-account and do not own a Savings-account.

$(\pi_{\text{c-owner}}\text{Checking-account}) - (\pi_{\text{s-owner}}\text{Savings-account})$

- What is the *schema* of result?

c-owner
W. Wei
M. Jones
H. Martin

Challenge Question

- How could you express the intersection operation if you didn't have an Intersection operator in relational algebra? [Hint: Can you express Intersection using only the Difference operator?]

$A \cap B =$???

Combining Tuples of Two Relations

Cross Product: Example

X cross product

Teacher

t-num	t-name
101	Smith
105	Jones
110	Fong

Teacher X Course

Course

c-num	c-name
514	Intro to DB
513	Intro to OS

	t-num	t-name	c-num	c-name
	101	Smith	514	Intro to DB
	105	Jones	514	Intro to DB
	110	Fong	514	Intro to DB
	101	Smith	513	Intro to OS
	105	Jones	513	Intro to OS
	110	Fong	513	Intro to OS

Cross product: combine information from 2 tables

- produces:

every possible combination of a teacher and a course

Cross Product

- R1 X R2
- Each row of R1 is paired with each row of R2.
- *Result schema* has one field per field of R1 and R2, with field names `inherited' if possible.

Renaming

- The ρ operator gives a new name to a relation.
- R1 := $\rho_{R1(A1,\ldots,An)}$(R2) makes R1 be a relation with attributes A1,…,A*n* and the same tuples as R2.
- Simplified notation: R1(A1,…,A*n*) := R2.

Example: Renaming

Bookstore(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

)

R(bs, addr) := Bookstore

R(

bs,	addr
Joe's	Maple St.
Sue's	River Rd.

)

Join: Example

⋈ = join

Account ⋈ $_{\text{Number=Account}}$ Deposit

Account	**Number**	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	**Account**	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00

	Number	Owner	Balance	Type	Account	Transaction-id	Date	Amount
	102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
	102	W. Wei	2000.00	checking	102	2	10/29/00	200.00
	104	M. Jones	1000.00	checking	104	3	10/29/00	1000.00
	105	H. Martin	10,000.00	checking	105	4	11/2/00	10000.00

Join: Example

⋈ join Account ⋈$_{\text{Number=Account}}$ Deposit

Note that when the join is based on equality, then we have two identical attributes (columns) in the answer.

Number	Owner	Balance	Type	Account	Trans-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00
104	M. Jones	1000.00	checking	104	3	10/29/00	1000.00
105	H. Martin	10,000.00	checking	105	4	11/2/00	10000.00

Join: Example

⋈ join Account ⋈ (Number=Account and Amount>700) Deposit

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

Number	Owner	Balance	Type	Account	T-id	Date	Amount
104	M. Jones	1000.00	checking	104	3	10/29/00	1000.00
105	H. Martin	10,000.00	checking	105	4	11/2/00	10000.00

Challenge Question (1)

- How could you express the "join" operation if you didn't have a join operator in relational algebra? [Hint: are there other operators that you could use, in combination?]

Joins

- *Condition Join*: $R \bowtie_c S = \sigma_c (R \times S)$
  - Sometimes called a *theta-join*

- *Result schema* same as that of cross-product
- Fewer tuples than cross-product, might be able to compute more efficiently

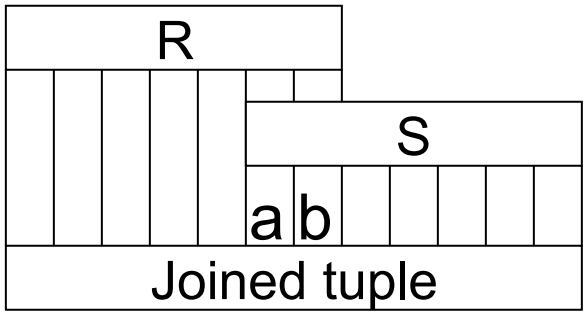
Challenge Question (2)

- What is an efficient way to evaluate this query?

$$\text{Account} \bowtie_{(\text{Number=Account and Amount>700})} \text{Deposit}$$

Natural Join

- *Join* on *all* common fields
  - R.a = S.a AND R.b=S.b
- *Result schema* similar to cross-product, but only one copy of fields for which equality is specified



Challenge Question

- How could you express the natural join operation if you didn't have a natural join operator in relational algebra? Consider you have two relations R(A,B,C) and S(B,C,D).

  ????