1

Module 03: Calculus

Björn Þór Jónsson

T-202-GAG1

Example Query

- Return drinkers that frequent all coffeehouses.

2 dálkar   1 dálkur

drinker, coffeehouse = öll kaffihús

$$\overline{F} \;/\; \Pi_{name \to coffeehouse}(H)$$

H := Coffeehouses(<u>name</u>, address, license)
F := Frequents(<u>drinker</u>, <u>coffeehouse</u>)

$A(x, y) / B(y)$

Example Query

- Return Drinkers who like all coffees.

$L \,/\, \pi_{name \to coffee}(Coffees)$

C := Coffee(<u>name</u>, manufacturer)
L := Likes(<u>drinker</u>, <u>coffee</u>)

$A(x, y) / B(y)$

Example Query

- Return drinkers that frequent all coffeehouses which sell Maracaibo.

$$F / \pi_{coffeehouse}(\sigma_{coffee='Mara..'}(S))$$

F := Frequents(drinker, coffeehouse)
S := Sells(coffeehouse, coffee, price)

$$A(x,y) / B(y)$$

Example Query

- Return coffeehouses that sell all coffees manufactured by Ottolina.

$$\pi_{coffeehouse,\,coffee}(S) \,/\, \pi_{name \to coffee}(\sigma_{manf='Ottolina'}(C))$$

$$C := Coffees(\underline{name}, manf)$$

~~H := Coffeehouses(name, address, license)~~

~~F := Frequents(drinker, coffeehouse)~~

S := Sells(coffeehouse, coffee, price)

6

Consider A(x, y) and B(y)

Find all x values in A that link with all y values in B

$A(NID, CID)$ $\quad B(CID)$

Calculating Division

- First find *all* existing x values

$$\pi_x(NID)$$

- Then find all the *possible* x, y combinations

$$\pi_{NID}(A) \times \pi_{CID}(B)$$

- Then remove all *existing* combinations

$$\pi_{NID}(A) \times B - A$$

- Find all x values that *do not link* with all y values in B

$$\pi_{NID}(\pi_{NID}(A) \times B - A)$$

- *Subtract* these from all x values in A

$$\pi_{NID}(A) - \pi_{NID}(\pi_{NID}(A) \times B - A)$$

7

Consider A(x, y) and B(y)

Find all x values in A that link with all y values in B

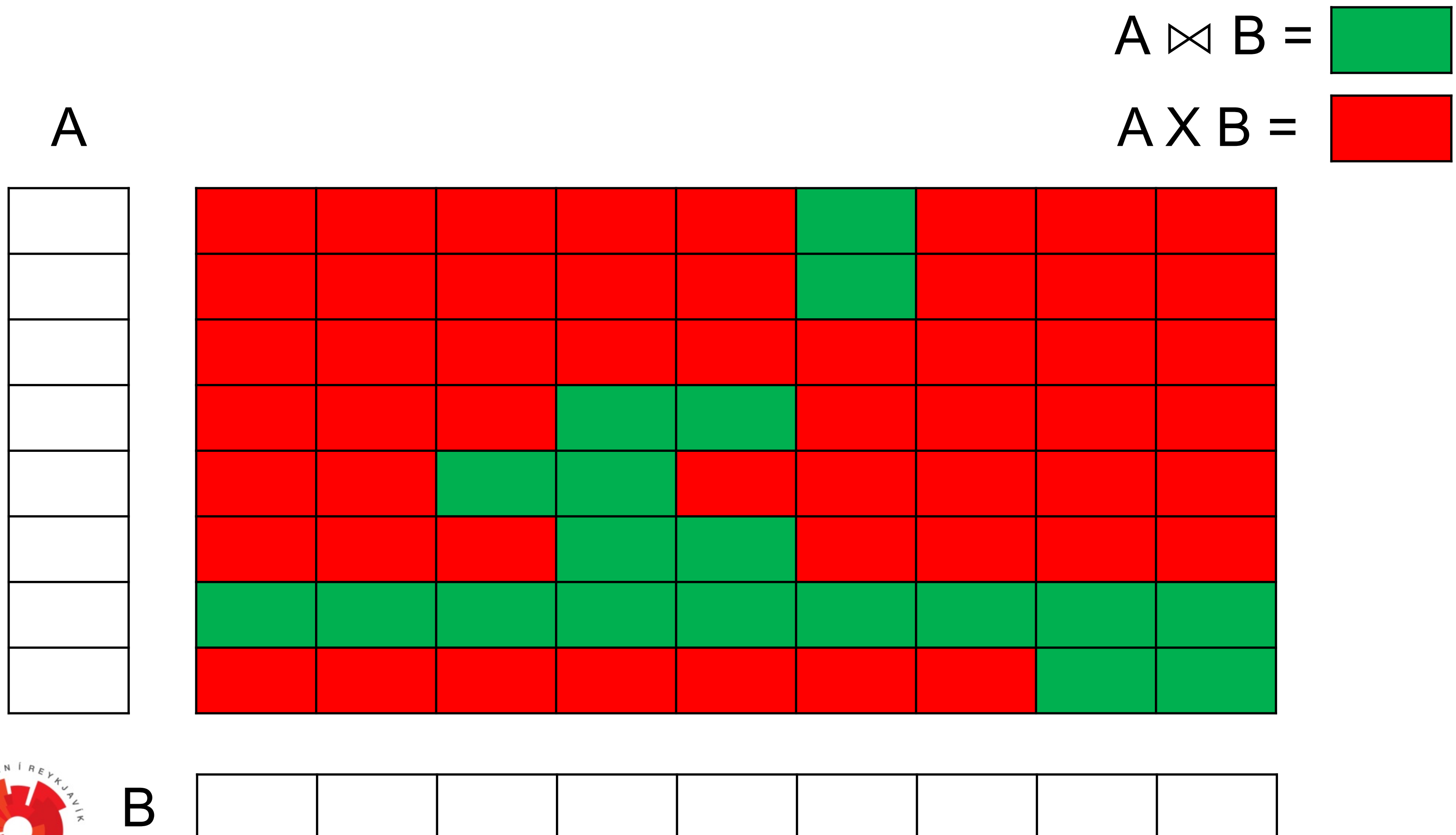
Graphically?

A X B =

Consider A(x, y) and B(y)

Find all x values in A that link with all y values in B

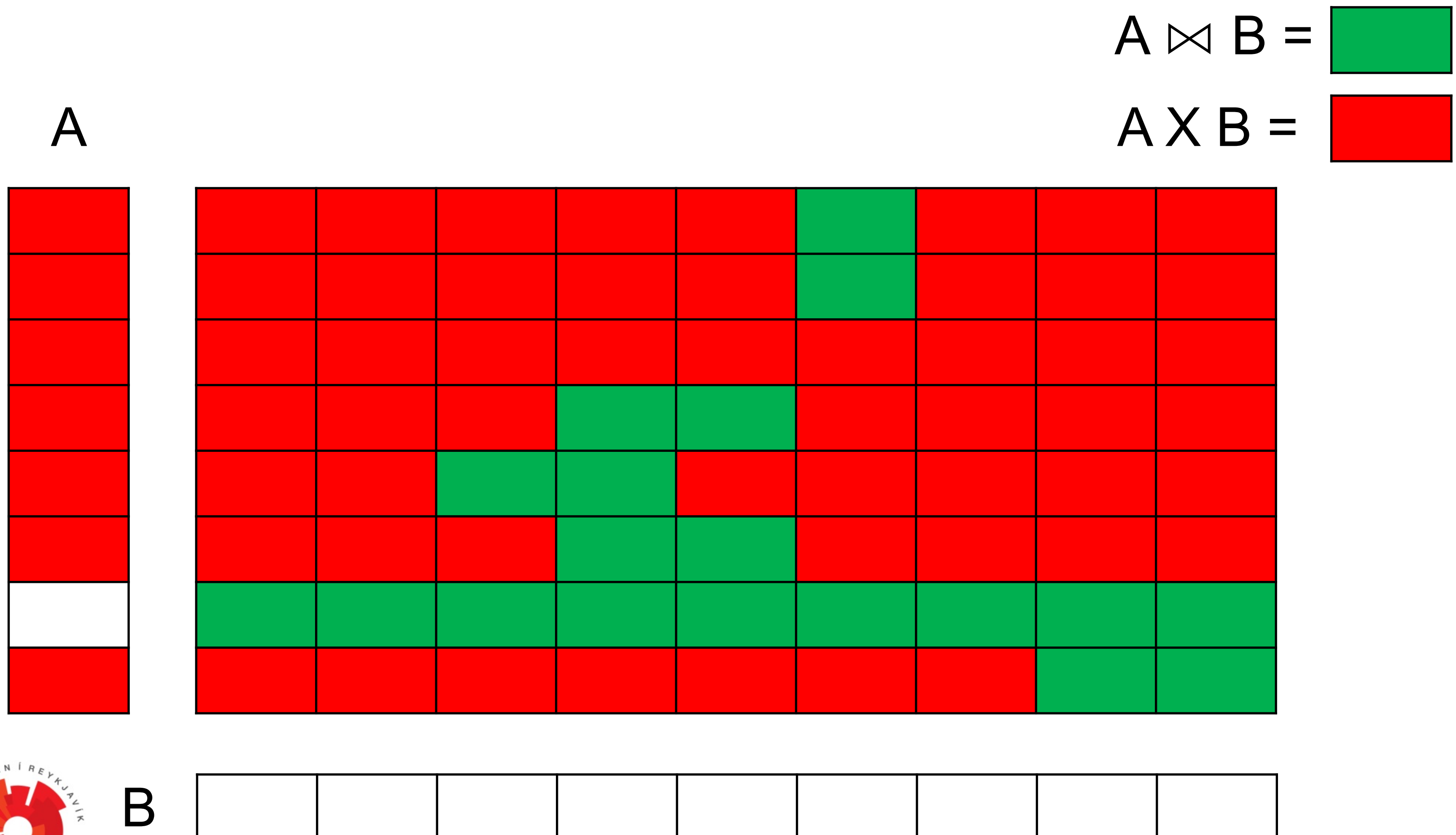
Graphically?



Consider A(x, y) and B(y)

Find all x values in A that link with all y values in B

Graphically?



Consider A(x, y) and B(y)

Find all x values in A that link with all y values in B

Graphically?

$A \bowtie B =$

$A \times B =$

A

B

Consider A(x, y) and B(y)

Find all x values in A that link with all y values in B

$A(NID, CID)$ $B(CID)$

Your Turn!

- First find *all* existing x values

$$\pi_{NID}(A)$$

- Then find all the *possible* x, y combinations

$$\pi_{NID}(A) \times B$$

- Then remove all *existing* combinations

$$\pi_{NID}(A) \times B - A$$

- Find all x values that *do not link* with all y values in B

$$\pi_{NID}(\pi_{NID}(A) \times B - A)$$

- *Subtract* these from all x values in A

$$\pi_{NID}(A) - \pi_{NID}(\pi_{NID}(A) \times B - A)$$

Example Query

- Return drinkers that frequent all coffeehouses, with and without division.

$A(\text{drinker}, \text{coffeehouse}) \; F \;/\; \pi_{\text{name} \to \text{coffeehouse}}(H)$

$B(\text{coffeehouse})$

$\text{VANTAR} := \pi_{\text{drinkers}}(F) \times \pi_{\text{name} \to \text{coffeehouse}}(H) - F$

$\pi_{\text{drinkers}}(F) - \pi_{\text{drinkers}}(\text{VANTAR})$

- First find *all* existing x values
- Then find all the *possible* x, y combinations
- Then remove all *existing* combinations
- Find all x values that *do not link* with all y values in B
- *Subtract* these from all x values in A

H := Coffeehouses(<u>name</u>, …)

F := Frequents(<u>drinker</u>, <u>coffeehouse</u>)

Module 02: Formal Semantics of SQL Queries

- Start with the product of all the relations in the FROM clause.
- Apply the selection condition from the WHERE clause.
- Project onto the list of attributes and expressions in the SELECT clause.

Building Complex Expressions

- Combine operators with parentheses and precedence rules
- Three notations, just as in arithmetic:
  1. Sequences of assignment statements
  2. Expressions with several operators
  3. Expression trees

Sequences of Assignments

- Create temporary relation names
- Renaming can be implied by giving relations a list of attributes
- Example: R3 := R1 $\bowtie_C$ R2 can be written:

  R4 := R1 X R2

  R3 := σ_C(R4)

Expressions in a Single Assignment

- Example: the theta-join R3 := R1 $\bowtie_C$ R2 can be written: R3 := σ_C (R1 X R2)
- Precedence of relational operators:
  - [σ, π, ρ] (highest).
  - [x, $\bowtie$].
  - $\cap$.
  - [$\cup$, —]

Example Query

- For all coffeehouses, show all coffees that are more expensive than some other coffee.
  - Hint: Use renaming to disambiguate coffee names and prices.

Example Query

- Using a sequence of assignments, return drinkers that frequent all coffeehouses, not using division.

- First find *all* existing x values
- Then find all the *possible* x, y combinations
- Then remove all *existing* combinations
- Find all x values that *do not link* with all y values in B
- *Subtract* these from all x values in A

H := Coffeehouses(<u>name</u>, …)
F := Frequents(<u>drinker</u>, <u>coffeehouse</u>)

Example Query

- Using a sequence of assignments, return drinkers that frequent all coffeehouses, using division.

H := Coffeehouses(<u>name</u>, …)
F := Frequents(<u>drinker</u>, <u>coffeehouse</u>)

Join Example

- Write this in algebra:

```
SELECT L.coffee
FROM Likes L
  JOIN Frequents F ON F.drinker = L.drinker
WHERE F.coffeehouse = 'Joe''s';
```

$$\pi_{coffee}\left(\sigma_{coffeehouse='J'}\left(F \bowtie L\right)\right)$$

Expression Trees

- Leaves are operands—either variables standing for relations or particular, constant relations
- Interior nodes are operators, applied to their child or children

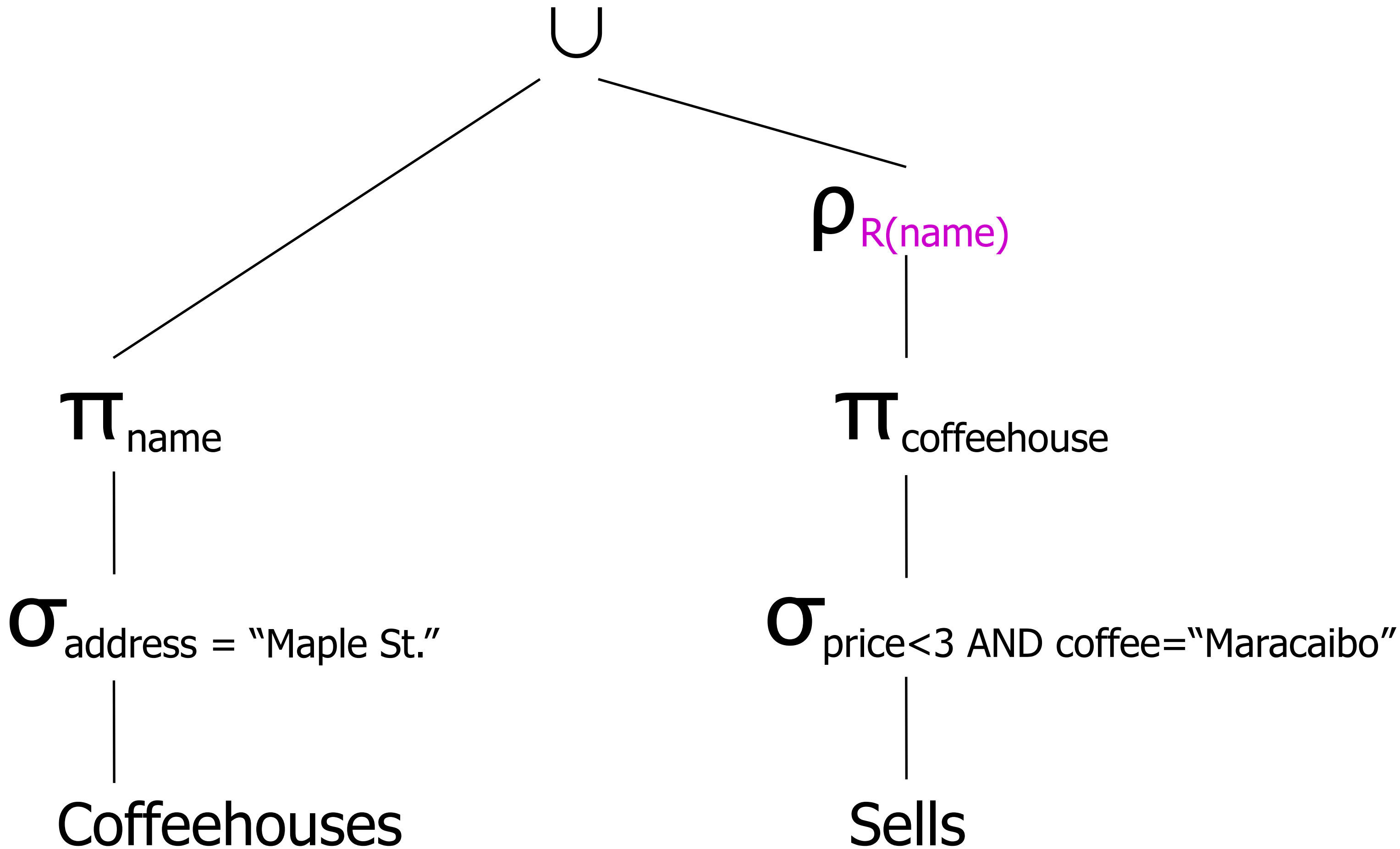
Example: Tree for a Query

- Using the relations Coffeehouses(name, address) and Sells(Coffeehouse, coffee, price), find the names of all the coffeehouses that are either on Maple St. or sell Maracaibo for less than $3

23

Using the relations Coffeehouses(name, address) and Sells(Coffeehouse, coffee, price), find the names of all the coffeehouses that are either on Maple St. or sell Maracaibo for less than $3

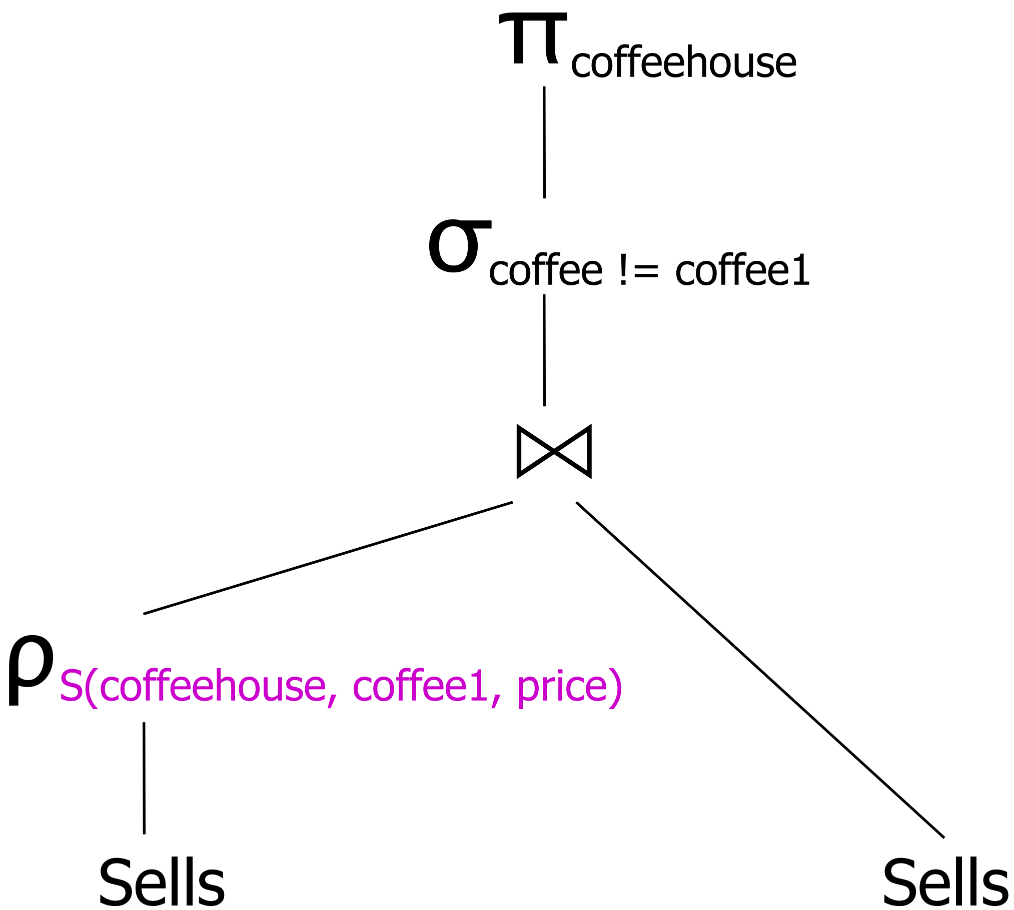
As a Tree



Example: Self-Join

- Using Sells(coffeehouse, coffee, price), find the coffeehouses that sell two different coffees at the same price
- Strategy: by renaming, define a copy of Sells, called S(coffeehouse, coffee1, price). The natural join of Sells and S consists of quadruples (coffeehouse, coffee, coffee1, price) such that the coffeehouse sells both coffees at this price

The Tree



$\pi_{coffeehouse}$
$\sigma_{coffee\ != \ coffee1}$
$\bowtie$
$\rho_{S(coffeehouse,\ coffee1,\ price)}$
Sells
Sells

Schemas for Results (1)

- Union, intersection, and difference: the schemas of the two operands must be the same, so use that schema for the result.
- Selection: schema of the result is the same as the schema of the operand.
- Projection: list of attributes tells us the schema.

A ∪ B

Schemas for Results (2)

- Product: schema is the attributes of both relations.
  - Use R.*A*, etc., to distinguish two attributes named *A*.
- Theta-join: same as product.
- Natural join: union of the attributes of the two relations.
- Renaming: the operator tells the schema.

Relational Algebra on Bags

- A *bag* (or *multiset*) is like a set, but an element may appear more than once
- Example: {1,2,1,3} is a bag
- Example: {1,2,3} is also a bag that happens to be a set

Why Bags?

- SQL, the most important query language for relational databases, is actually a bag language
- Some operations, like projection, are more efficient on bags than sets

Operations on Bags

- Selection applies to each tuple, so its effect on bags is like its effect on sets
- Projection also applies to each tuple, but as a bag operator, we do not eliminate duplicates
- Products and joins are done on each pair of tuples, so duplicates in bags have no effect on how we operate

Example: Bag Selection

R(A, B)

	A,	B
1	2	
5	6	
1	2	

$\sigma_{A+B<5}(R) =$

	A	B
1	2	
1	2	

Example: Bag Projection

R(

A,	B)
1	2
5	6
1	2

$\pi_A(R) =$

A
1
5
1

Example: Bag Product

R(A, B)

A	B
1	2
5	6
1	2

S(B, C)

B	C
3	4
7	8

R X S =

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	3	4
5	6	7	8
1	2	3	4
1	2	7	8

Example: Bag Theta-Join

R(A, B)

	A,	B
	1	2
	5	6
1	2	

S(B, C)

B,	C
3	4
7	8

$R \bowtie_{R.B<S.B} S =$

	A	R.B	S.B	C
1	2	3	4	
1	2	7	8	
5	6	7	8	
1	2	3	4	
1	2	7	8	

Bag Union

- An element appears in the union of two bags the sum of the number of times it appears in each bag
- Example: $\{1,2,1\} \cup \{1,1,2,3,1\} = \{1,1,1,1,1,2,2,3\}$

$\{1,2\} \cup \{1,2,3\} = \{1,2,3\}$

Bag Intersection

- An element appears in the intersection of two bags the minimum of the number of times it appears in either
- Example: $\{\cancel{1},2,1,1\} \cap \{1,2,1,3\} = \{1,1,2\}$

Bag Difference

- An element appears in the difference $A - B$ of bags as many times as it appears in A, minus the number of times it appears in B
  - But never less than 0 times
- Example: {1,2,1,1} – {1,2,3} = {1,1}

$3 - 1 = 2$

Beware: Bag Laws != Set Laws

- Some—but *not all*—algebraic laws that hold for sets also hold for bags
- Example: the commutative law for union ($R \cup S = S \cup R$) *does* hold for bags
  - Since addition is commutative, adding the number of times x appears in R and S doesn't depend on the order of R and S

Example: A Law That Fails

- Set union is *idempotent*, meaning that $S \cup S = S$
- However, for bags, if x appears n times in S, then it appears $2n$ times in $S \cup S$
- Thus $S \cup S$!= S in general
  - e.g., {1} $\cup$ {1} = {1,1} != {1}

Take-away Points

- Relational algebra:
  - Operands are relations (sets of tuples with associated schema)
  - Operations are:
    - Select, project, cartesian product, set difference, union
    - Joins (thetajoin, equijoin, natural join, outer join)
    - Intersection, division
  - Used for query optimization and processing

Part B: Tuple Relational Calculus

Björn Þór Jónsson

T-202-GAG1

Tuple Relational Calculus

- A declarative (non-procedural) query language
  - What?
  - Not: How?
- Queries: { R | p(R) }
  - R is a tuple variable
  - p(R) is a formula that defines R

Syntax of TRC

- Atomic formulas
  - Bind variables: $R \in$ Relation $D \in$ Drinker
  - Compare attributes to attributes $R.a = S.b$ $\neq$ $<$
  - Compare attributes to constants $R.c = 5$
- Formulas
  - Atomic formulas
  - Logical connectives p $\neg p$ $p \vee q$ $p \wedge q$ $p \Rightarrow q$
  - Existential quantifier $\exists F \in$ Frequents( )
  - Universal quantifier $\forall F \in$ Frequents( )

Selection

- Info about Joe's

$\{R \mid R \in Coffeehouse \wedge R.name = \text{'Joe's'}\}$

Projection

- Address of Joe's

$$\{R \mid \exists H \in \text{Coffeehouses}(H.\text{name} = \text{'Joe's'} \wedge R.\text{address} = H.\text{address})\}$$

FROM

WHERE

SELECT

Example Query

- Coffees sold at Joe's for less than $2

FROM

$$\{ R \mid \exists S \in Sells (S.coffeehouse = \text{'Joe's'} \land S.price < 2 \land R.coffee = S.coffee) \}$$

WHERE

SELECT

Joins

- Name of drinkers who frequent Joe's

$$\{R \mid \exists F \in Frequents\ (F.coffeehouse = \text{'Joe's'} \wedge R.name = F.drinker)\}$$

- Address of drinkers who frequent Joe's

$$\{R \mid \exists F \in Frequents\ \exists D \in Drinkers\ (F.drinker = D.name \wedge F.coffeehouse = \text{'Joe's'} \wedge R.address = D.address)\}$$

Example Query

- Names and addresses of coffeehouses that sell Maracaibo for less than $1

$\{R \mid \exists H \in Coffeehouse.\ \exists S \in Sells($
$H.name = S.coffeehouse \wedge$
$S.coffee = \text{'Maracaibo'} \wedge$
$S.price < 1 \wedge$
$R.name = H.name \wedge$
$R.address = H.address)\}$

WHERE

SELECT

FROM Coffeehouse H JOIN Sells S
ON H.name = S.coffeehouse

Example Query

- For any drinker who frequent any coffeehouse, addresses of the drinker (daddr) and the coffeehouse (caddr)

$$\{R \mid \exists D \in Drinker\ \exists H \in Coffeehouse\ \exists F \in Frequent(\ D.name = F.drinker \wedge H.name = F.coffeehouse \wedge R.daddr = D.address \wedge R.caddr = H.address)\}$$

Set Operations

- Name of drinkers who don't frequent Joe's

$$\{ R \mid \exists D \in \text{Drinkers} (R.\text{name} = D.\text{name}) \}$$

$$-$$

$$\{ R \mid \exists F \in \text{Frequents} (F.\text{coffeehouse} = \text{'Joe's'} \wedge R.\text{name} = F.\text{drinker}) \}$$

"ALL" Join

Division

BIRTA

- Names of coffeehouses that sell all coffees

$\{R \mid \exists H \in Coffeehouse\ \forall C \in Coffees\ \exists S \in Sells($

"Join" | $H.name = S.coffeehouse \wedge$

$C.name = S.coffee \wedge$

SELECT → $R.name = H.name)\}$

Example Query

?

- Names of drinkers who frequent all coffeehouses

$$\{R \mid \exists D \in Drinkers\ \forall H \in Coffeehouses\ \exists F \in Frequents\ (D.name = F.drinker \wedge H.name = F.coffeehouse \wedge R.name = D.name)\}$$

Example Query

- Names of drinkers who like all coffees

Division with Double Negation

- Use a property of quanitifiers
  - For all x there exists y = There is no x such that there is no y

$$\forall x \exists y \equiv \neg \exists x \neg \exists y$$

$$\{R \mid \exists D \in Drinkers(\neg \exists H \in Coffeehouses(\neg \exists F \in Frequents($$

Division with Double Negation

- Names of coffeehouses that sell all coffees
  - All coffeehouses H such that there is no coffee C such that there is no tuple in Sells for H and C

$$\{R \mid \exists H \in Coffeehouses\ \neg\exists C \in Coffees\ \neg\exists S \in Sells\ (H.name = S.coffeehouse \wedge C.name = S.coffee \wedge R.name = H.name)\}$$

Division with Double Negation Translated to SQL

- Names of coffeehouses that sell all coffees
  - All coffeehouses H such that there is no coffee C such that there is no tuple in Sells for H and C

```
Select H.name
From Coffeehouses H
Where not exist (
  Select *
  From Coffees C
  WHERE NOT EXISTS (
    SELECT *
    FROM Sells S
    WHERE C.name = S.coffee
    AND H.name = S.coffeehouse))
```

Example Query

- Names of drinkers who frequent all coffeehouses
  - All drinkers D such that there is no coffeehouse H such that there is no tuple in Frequents for D and H

Example Query

- Names of drinkers who like all coffees
  - All drinkers D such that there is no coffee C such that there is no tuple in Likes for D and C

Take-away Points

- Relational algebra:
  - Operands are relations (sets of tuples with associated schema)
  - Operations are:
    - Select, project, cartesian product, set difference, union
    - Joins (thetajoin, equijoin, natural join, outer join)
    - Intersection, division
  - Used for query optimization and processing
- Tuple Relational Calculus:
  - Formulas that define sets of tuples
    - Quantifiers, logical connectives, …
    - Can express anything that algebra can express
  - Foundation for SQL