Unit 5
Formal Relational Languages

Text: Chapters 4 & 24

Relational Algebra (Ch. 4: 4.1—4.2)

Tuple Relational Calculus X

Domain Relational Calculus X

Datalog (Ch. 24)

Learning Goals

Given a database (a set of tables) you will be able to

- express a database query in Relational Algebra, involving the basic operators (selection, projection, cross product, renaming, set union, intersection, difference), join, division and assignment
- rewrite RA expressions (queries) using a subset of the operators with expressions using another subset
- show that two RA queries are/aren't equivalent
- express a DB query in Datalog
- translate RA queries to Datalog; translate queries from a fragment of Datalog to RA.

Formal Relational Query Languages

- Mathematical Languages that form the basis for the implementation of "real" languages (e.g., SQL
- *Relational Algebra*: More procedural, very useful for representing query evaluation plans.
- *Relational Calculi (Tuple or Domain)*: Let users describe what they want, rather than how to compute it. (Non-procedural, ***declarative***.) X
- Datalog: Declarative; helpful in writing RA and SQL queries correctly.

  - *Understanding Algebra & Datalog is key to understanding SQL, query processing*

Relational Algebra (RA)

- Procedural language
- Basic operations:
  - ***Selection*** - Selects a subset of rows from relation.
  - ***Projection*** - 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.
  - ***Rename*** – Assigns a(nother) name to a relation
- Additional operations:
  - ***intersection, join, division, assignment***: not essential, but very useful
- The operators take one or two relations as inputs and give a new relation as a result.
- Operations can be ***composed***. (Algebra is "closed".)

Datalog (Lite)

- Will see Datalog (full) later.
- Rule-based: **head** ← **body**.
- **head** is of the form $p(X_1, \ldots, X_n)$. Here, X_i are **variables**. Think of $p(\ldots)$ as a "collector" of answers to the query expressed by the **body**.
- **body** is a conjunction (i.e., AND) of atoms of two kinds:
  - $r(Y_1, \ldots, Y_k)$ where r is a database relation.
  - $X\ op\ c$ or $X\ op\ Y$ where X, Y are variables and c is a constant.
- Variables are bound to values appearing in the DB when query is evaluated.
- Constants can be numerical, string, etc.

Example Instances

- Instances of "Customer" and "Order" relations for our running example of online store.

Customer

cid	cname	rating	salary
32	G. Grumpy	5	95
51	S. Sneezy	8	55
68	R. Rusty	10	55

Order

cid	iid	day	qty
32	101	30/05/07	10
68	103	10/12/06	5

Selection

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

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

Set of tuples of r that satisfy p

Where *p* is a formula in propositional calculus consisting of

predicates

connectives : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)

A **predicate** is one of:

<attribute> *op* <attribute> or

<attribute> *op* <constant>

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

❖ Result schema is same as r's schema

Selection Example 1

- Relation *r*

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

RA

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

Datalog

$$ans(A, B, C, D) \leftarrow r(A, B, C, D), A = B, D > 5$$

variable

attribute

constant

A	B	C	D
α	α	1	7
β	β	23	10

Selection Example 2

cust

cid	cname	rating	salary
21	Y. Yuppy	5	95
50	B. Rusty	10	65
55	S. Sneezy	8	70

$\sigma_{rating>7}(cust)$

$ans(I, C, R, S) \leftarrow cust(I, C, R, S), R > 7.$

cid	cname	rating	salary
50	R. Rusty	10	65
55	S. Sneezy	8	70

Selection Example 3

Cust

cid	cname	rating	salary
21	Y. Yuppy	5	95
50	B. Rusty	10	65
55	S. Sneezy	8	70

- Find customers in Cust with rating less than 9 and salary more than 80.

Projection

- Notation:

$$\pi_{A_1, A_2, \ldots, A_k}(r)$$

  where $A_1, \ldots, A_k$ are attributes (the projection list) and r is a relation.
- The result = relation over the k attributes $A_1, A_2, \ldots, A_k$ obtained from r by erasing the columns that are not listed and **eliminating duplicate rows**.
- Remember: relations are sets!

Projection Example 1

- Relation r:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$ans(A, C) \leftarrow r(A, B, C).$

Or you may write
$ans(X, Z) \leftarrow r(X, Y, Z).$

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

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Variable name not important.

Projection Example 2

$\pi_{salary}(Cust)$

salary
95
55

Cust

cid	cname	rating	salary
38	R. Rudy	9	95
32	G. Grumpy	8	55
51	S. Sneezy	5	95
78	R. Rusty	10	55

$\pi_{cname,rating}(Cust)$

cname	rating
R. Rudy	9
G. Grumpy	8
S. Sneezy	5
R. Rusty	10

$\pi_{cname,rating}(\sigma_{rating>7}(Cust))$?

$ans(C, R) \leftarrow cust(I, C, R, S), R > 7.$

Union, Intersection, Set-Difference

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

  $r \cup s = \{t \mid t \in r \textbf{ or } t \in s\}$
  $r \cap s = \{t \mid t \in r \textbf{ and } t \in s\}$
  $r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$
- For these operations to be well-defined:
  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* has same domain of values as the 2nd column of *s*)
- What is the schema of the result?

Union,Int., Diff. Examples

Relations r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

Where did we step outside Datalog lite?

$ans(X,Y) \leftarrow r(X,Y).$
$ans(X,Y) \leftarrow s(X,Y).$

$ans(X,Y) \leftarrow r(X,Y), s(X,Y).$

r ∪ s:

A	B
α	1
α	2
β	1
β	3

r ∩ s:

A	B
α	2

$r - s$:

A	B
α	1
β	1

$ans(X,Y) \leftarrow r(X,Y), \neg s(X,Y).$

Union,Int., Diff. Examples

*C*1

cid	cname	rating	salary
22	J. Justin	7	65
31	R. Rubber	8	85
58	N. Nusty	10	85

*C*2

cid	cname	rating	salary
28	Y. Yuppy	9	95
31	R. Rubber	8	85
44	G. Guppy	5	70
58	N. Nusty	10	85

*C*1∪*C*2

cid	cname	rating	salary
22	J. Justin	7	65
31	R. Rubber	8	85
58	N. Nusty	10	85
44	G. Guppy	5	70
28	Y. Yuppy	9	95

*C*1–*C*2

cid	cname	rating	salary
58	J. Justin	7	65

*C*1∩*C*2

cid	cname	rating	salary
31	R.Rubber	8	85
58	N. Nusty	10	85

How do you write these queries in Datalog?

Cartesian (or Cross)-Product

- 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, $R \cap S = \varnothing$).
- If r and s have common attributes, they must be renamed in the result.

Cartesian-Product Example 1

r

A	B
α	1
β	2

s

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

$ans(X, Y, Z, W, U) \leftarrow r(X, Y), s(Z, W, U).$

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

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

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

$ans(X, Y, Z, W, U) \leftarrow r(X, Y), s(Z, W, U), X = Z.$

$ans(X, Y, X, W, U) \leftarrow r(X, Y), s(X, W, U).$

Cartesian-Product Example 2

Customer

cid	cname	rating	salary
22	J. Justin	7	65
31	R. Rubber	8	85
58	N. Nusty	10	85

Order

cid	iid	day	qty
22	101	10/10/06	10
58	103	11/12/06	5

conflicting names

Customer x Order

Customer .cid	sname	rating	salary	Order. cid	iid	day	qty
22	J. Justin	7	65	22	101	10/10/96	10
22	J. Justin	7	65	58	103	11/12/96	5
31	R. Rubber	8	85	22	101	10/10/96	10
31	R. Rubber	8	85	58	103	11/12/96	5
58	N. Nusty	10	85	22	101	10/10/96	10
58	N. Nusty	10	85	58	103	11/12/96	5

Attribute names present no issues for Datalog.

Rename

- Allows us to name results of relational-algebra expressions.
- Allows us to assign more names to a relation .
- Allows us to rename attributes of a relation.
- Notation

  $\rho_x(E)$

  returns the expression E under the name X
- If 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 $A1, A2, \ldots, An$.
- $\rho_{X(B1 \to A1, \ldots, Bk \to Ak)}(E)$

  is as before, but it only renames attributes B1,…,Bk of E to A1,…,Ak.

Rename

- Not much in the way of Rename can be done in Datalog, and there is no need to.
- Recall, Datalog ignores attribute names and instead focuses on their position.
- This means, by not having a feature for Rename, Datalog doesn't lose any expressive power: you can express all RA queries in Datalog.
- However, since Datalog is a rule-based language, you naturally get to name the (relation) collector that holds answers to (intermediate) queries.
- Will next see advanced features in RA and Datalog. But first we will take a look at …

Rename Example

relation renaming

- cust(cid, cname, rating, salary).
- Find pairs of customer names (c1,c2) such that c1 is rated higher than c2 but is paid less.

In RA: $\pi_{cname,cust1.cname}(\sigma_{rating>cust1.rating \wedge salary<cust1.salary}(cust \times \rho_{cust1}(cust)))$.

Attr renaming

$\pi_{cname,cname'}(\sigma_{rating>rating' \wedge salary<salary'}(cust \times \rho_{cid \to cid',cname \to cname',rating \to rating',salary \to salary'}(cust)))$.

In Datalog: What (re)name? ☺

ans(C,C') ← *cust(I,C,R,S), cust(J,C',R',S'), R>R', S<S'.*

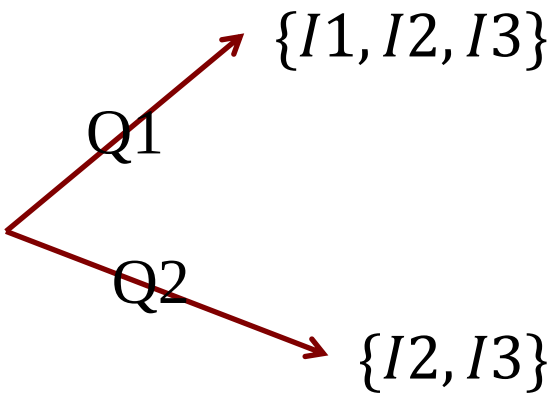
Different Shades of Negation

$\neq$ versus $\neg$

- Recall the relations cust(cid, cname, rating, salary) and ord(cid, iid, day, qty) and consider the queries:
- Q1: Find items (iid) ordered by someone other than the customer with cid 32.
- Q2: Find items in *ord* that are not ordered by customer with cid 32.

cid	iid	day	qty
32	I1	15/01/2013	5
23	I1	16/01/2013	3
23	I2	17/01/2013	2
16	I3	15/01/2013	2

An instance of *ord*



Diff. Shades of Negation (contd.)

- We can express Q1 as $\pi_{iid}(\sigma_{cid \neq 32}(ord))$ in RA.
- In Datalog, we can write this query as $ans(I) \leftarrow ord(C, \mathrm{I}, \mathrm{D}, \mathrm{Q}), \mathrm{C} \neq 32.$ ord(cid, iid, day, qty)
- We can express Q2 as $\pi_{iid}(ord) - \pi_{iid}(\sigma_{cid=32}(ord))$.
- We can write this query using 2 rules in Datalog:
  - $bad(I) \leftarrow ord(32, I, D, Q).$
  - $good(I) \leftarrow ord(C, I, D, Q), \neg bad(I).$

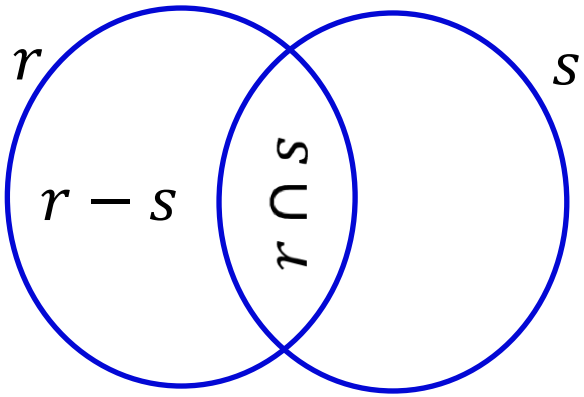
We've stepped well outside Datalog Lite. IOW, we've started seeing Datalog Full. What new features are we using?

Additional Operations

- They can be defined in terms of the primitive operations
- They are added for convenience
- They are:
  - Set intersection (we've seen it)
  - Join (Condition, Equi-, Natural)
  - Division
  - Assignment

Set intersection in terms of minus

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



Joins

Join: One of the most important ops implemented in a DBMS. Many efficient algorithms.

- *Condition Join*:

$$R \bowtie_C S = \sigma_C (R \times S)$$

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product
  - might be able to compute more efficiently
- Sometimes called a *theta-join*.

Condition Join Example

C1

cid	cname	rating	salary
22	J. Justin	7	80
31	R. Rubber	8	70
58	N. Nusty	10	90

O1

cid	iid	day	qty
22	101	10/10/96	10
58	103	11/12/96	5

$$C1 \bowtie_{C1.cid < O1.cid} O1$$

C1.cid	cname	rating	salary	O1.cid	iid	day	qty
22	J. Justin	7	80	58	103	11/12/96	5
31	R. Rubber	8	70	58	103	11/12/96	5

Equi-Join & Natural Join

- *Equi-Join*: A special case of condition join where the condition *c* contains only ***equalities***
  - *Result schema:* similar to cross-product, but contains only one copy of fields for which equality is specified
- *Natural Join*: Equijoin on *all* common attrs.
  - *Result schema:* similar to cross-product, but contains only one copy of each common field
  - no need to show the condition
  - what if relations have no common attrs?

Equi & Natural Join Examples

O1

cid	iid	day	qty
22	101	10/10/96	10
58	103	11/12/96	5

C1

cid	cname	rating	salary
22	J. Justin	7	85
31	R. Rubber	8	95
58	N. Nusty	10	90

$C1 \bowtie_{C1.cid = O1.cid} O1$

cid	cname	rating	salary	iid	day	qty
22	J. Justin	7	85	101	10/10/96	10
58	N. Nusty	10	90	103	11/12/96	5

$C1 \bowtie O1$

cid	cname	rating	salary	iid	day	qty
22	J. Justin	7	85	101	10/10/96	10
58	N. Nusty	10	90	103	11/12/96	5

Equi & Natural Join conclusion

- Both produce relations that are essentially equivalent, i.e., have same content.
- Differences – superficial.
- Datalog does not distinguish between diff. kinds of joins.
- Datalog syntax makes intent explicit.
- A relational DBMS almost never implements $\times$ directly: inefficient and not natural.

$$equi(C, N, R, S, I, D, Q, C) \leftarrow cust(C, N, R, S), ord(C, I, D, Q).$$

$$nat(C, N, R, S, I, D, Q) \leftarrow cust(C, N, R, S), ord(C, I, D, Q).$$

Division

- Notation: r / s or $r \div s$
- Useful for expressing queries that include a "**for all**" or "**for every**" phrase
- Let r and s be relations on schemas R and S respectively where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

  Then r/s is a relation on schema

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

  defined as

  $$r / s = \{ t \mid t \in \prod_{R-S}(r) \wedge \forall u \in s (tu \in r) \}$$
- Informally, r / s contains the (parts of) tuples of r that are associated with every tuple in s.

Examples of Division A/B

A

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

A/B1

sno
s1
s2
s3
s4

B2

pno
p2
p4

A/B2

sno
s1
s4

B3

pno
p1
p2
p4

A/B3

sno
s1

More on Division

Recall: **cust(cid,cname,rating,salary),**
ord(cid,iid,day,qty).

Query: Find items (iid) that are ordered by **every** customer.

Don't know beforehand how many customers there are.

If there are 5 customers and you know their cid's (or look them up), how will you write this query in RA? What if there are 100?

Division lets us write this query concisely no matter how many customers …

Division (contd.)

Notice the projections!
Notice the order of attrs!

- $\pi_{iid,\ cid}(ord) \div \pi_{cid}(cust)$.
- What is this expression really doing? Wearing the Datalog hat answers this question! ☺
- $good(I) \leftarrow ord(\_, I, \_, \_), \neg bad(I).$

  $bad(I) \leftarrow cust(C, \_, \_, \_), ord(\_, I, \_, \_), \neg witness(C, I).$

  $witness(C, I) \leftarrow ord(C, I, \_, \_).$
- Query answer = the set of "good" items.
- In RA, using only basic ops:

  $\pi_{iid}(ord) - \pi_{iid}((\pi_{cid}(cust) \times \pi_{iid}(ord)) - \pi_{cid,iid}(ord))$.
- Notice the correspondence between double negation and double minus. Make sure to understand why we need double minus (negation) for this query!
- Notice type compatibility: only items are being subtracted from items.

Expressing r÷s Using Basic Operators

- Generalizing from previous example …
- To express r÷s think as
- *Idea*:
  - let X = R-S (X is the set of attributes of R that are not in S)
  - (1) compute the X-projection of r
  - (2) compute all *X*-projection values of r that are `disqualified' by some value in *s*.
    - value *x* is *disqualified* if by attaching *y* value from *s*, we obtain an *xy* tuple that is not in *r*.
  - result is (1)-(2)
- So,
  - Disqualified *x* values: $\pi_X((\pi_X(r)\times s)-r)$
  - r ÷ s is $\pi_X(r)-\pi_X((\pi_X(r)\times s)-r)$

Life beyond Division

- Consider expert(eid,ename,expertise,salary), item(iid,iname,area), and certify(eid,iid,day).
- Attrs expertise and area have same domain: e.g., items can be of type electronics and a customer may have expertise in electronics.
- Query: find those items that have been certified by all experts in the area of the item.
- Problem with division: no fixed divisor – divisor depends on item!
- No additional op we have studied can express this query directly! ☹
- Need to fall back on basic ops.

An Example

item

iid	…	area
i1	…	photography
i2	…	photography
i3	…	gardening
i4	…	gardening

expert

eid	…	expertise …
e1	…	photography
e2	…	photography
e3	…	gardening
e4	…	gardening

certify

eid	iid	…
e1	i1	…
e2	i1	…
e1	i3	…

certify

eid	iid	…
e3	i3	…
e4	i4	…
e3	i4	…

Life beyond division (conclusion)

- Don't despair: wear datalog hat:
- $good(I) \leftarrow item(I, \_, \_), \neg bad(I).$

$bad(I) \leftarrow$
$item(I, \_, A), expert(E, \_, A, \_), \neg witness(E, I).$
$witness(E, I) \leftarrow certify(E, I, \_).$

- In RA, this becomes:

  $\pi_{iid}(item) -$
  $\pi_{iid}(\pi_{eid,iid}(\sigma_{expertise=area}(expert \times item)) -$
  $\pi_{eid,iid}(certify)$

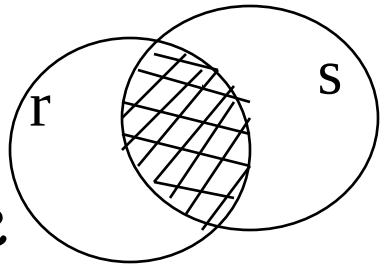
- Notice similarity in structure to expression for division, in particular double minus (negation).

Assignment Operation

- Notation: temp $\leftarrow$ E
  assigns the result of expression E to a temporary relation temp.
- Used to break complex queries to small steps.
- Assignment is always made to a temporary relation variable.
- Example1: Write $r \cap s$ in terms of $\cup$ and -

  $temp1 \leftarrow r - s$

  $r - temp1$



- Example2: revisit division and the previous example, using assignment.

Example: Customer Database

We use the following database:

Customer(*cid*: integer, *cname*: string, *rating*: integer, *salary*: real)

Item(*iid*: integer, *iname*: string, *type*: string)

Order(*cid*: integer, *iid*: integer, *day*:date, *qty*:real)

Find names of customers who've ordered item with id 100

- Solution 1:

$$\pi_{cname}(\sigma_{iid=100}(Order) \bowtie Customer)$$

- Solution 2:

$$temp1 \leftarrow \sigma_{iid=100}(Order)$$
$$temp2 \leftarrow temp1 \bowtie Customer$$
$$\pi_{cname}(temp2)$$

- Solution 3:

$$\pi_{cname}(\sigma_{iid=100}(Order \bowtie Customer))$$

Find names of Customers who've ordered a laptop (i.e., an item of type "laptop")

- Information about item type is only available in Item; so need an extra join:

$$\pi_{cname}(\sigma_{type='laptop'}(Item) \bowtie Order \bowtie Customer)$$

- A more efficient solution:

$$\pi_{cname}(\pi_{cid}((\pi_{iid}(\sigma_{type='laptop'}(Item)) \bowtie Order) \bowtie Customer)$$

✉ *A query optimizer can find this given the first solution!*

Find customers who've ordered a laptop or a desktop computer

- Can identify all laptops and desktops, then find Customers who've ordered one of these items:

$$temp \leftarrow \sigma_{type='laptop' \vee type='desktop'}(Item))$$

$$\pi_{cname}(temp \bowtie Order \bowtie Customer)$$

- Can also define *temp* using union! (How?)
- What happens if $\vee$ is replaced by $\wedge$ in this query?

Find customers who've ordered a laptop and a desktop computer

- Previous approach won't work! Must identify customers who've ordered laptops, customer who've ordered desktops, then find the intersection (note that *cid* is a key for Customer):

$$claps \leftarrow \pi_{cid}(\sigma_{type='laptop'}(Item) \bowtie Order))$$

$$cdesks \leftarrow \pi_{cid}(\sigma_{type='desktop'}(Item) \bowtie Order))$$

$$\pi_{cname}((claps \cap cdesks) \bowtie Customer)$$

Find the names of customers who've ordered all items

- Uses division; schemas of the input relations *must be carefully chosen*:

$$temp \leftarrow \pi_{cid,iid}(Order) \div \pi_{iid}(Item)$$

$$\pi_{cname}(temp \bowtie Customer)$$

- To find customers who've ordered all 'laptop' items:

..... $\div \pi_{iid}(\sigma_{type='laptop'}(Item))$

5

Safety of Datalog Expressions

- Some expressions in Datalog are not safe as they can generate infinite relations.
- E.g.: $ans(I) \leftarrow \neg item(I, N, T)$.
  $ans(I, C) \leftarrow item(I, \_, T), T \neq `hardware'$.
  $ans(I, C) \leftarrow item(I, \_, \_), C \neq 123$.
- To avoid the problem, we restrict the set of allowable expressions to safe expressions.
- **Roughly speaking**, a datalog rule is *safe* if if each variable in the head appears positively in one of the relations, or is bound by constants that appear in *the rule.*
- A datalog query is safe if all rules that make it up are.
- A relational algebra query can always be expressed as a safe Datalog; the converse is also true.

Summary

- Relational algebra is a procedural language providing a set of operations on relations. A query is expressed as a sequence of these operations.
- RA is complete in the sense that any "reasonable" query can be expressed in it.
- RA is used by commercial languages to define query evaluation plans.
- Datalog is a declarative language based on logic.
- We mainly discussed non-recursive Datalog.
- In general, Datalog is more powerful than RA: Datalog can express recursive queries; RA together with assignment and loop can express such queries.