The Expressiveness of Relational Algebra

From Chapter 2 of

Atzeni & De Antonellis, *Relational Database Theory*,

Benjamin/Cummings, 1993.

Basic Notions

- assume a countably infinite set of attributes U_∞
- assume all attributes have the same domain D.
- $\boldsymbol{U}$: set of all relations over every finite subset of U_∞
- $I(\mathbf{R})$: set of all instances of db schema $\mathbf{R}$
- Query $Q : I(\mathbf{R}) \rightarrow \boldsymbol{U}$
- $\mathbf{r} \in I(\mathbf{R})$: $Q(\mathbf{r})$ is the result of Q on $\mathbf{r}$
- X: set of attributes over which result of Q is defined

  $\boldsymbol{X}$: set of all relations on X

  Then $Q: I(\mathbf{R}) \rightarrow \boldsymbol{X}$

- Relational Query Languages
  - SQL, Quel, QBE
  - Relational Algebra, Relational Calculus

- Queries are expressed by *expressions* in some query language
  - Syntax: characterizes the legal expressions
  - Semantics: the value of $E(\mathbf{r})$ when E is a valid (legal) expression

- E represents Q if the function defined by E is indeed Q

- E_1, E_2 are equivalent with respect to **R** ($E_1 \equiv_{\mathbf{R}} E_2$)

  if they represent the same query

  i.e., if $E_1(\mathbf{r}) = E_2(\mathbf{r})$ for every $\mathbf{r} \in I(\mathbf{R})$

  $E_1 \equiv E_2$

  if E_1, E_2 are defined over same set of database schemes and equivalent w.r.t. each of them

- Languages L_1 and L_2
  - L_1 is at least as expressive as L_2
    if for every expression E_2 of L_2,
    there is an expression E_1 of L_1 s.t. $E_1 \equiv E_2$
  - L_1 is equivalent to L_2 if each is at least as expressive as the other

*** Review ***

- function f: $A \rightarrow B$ is *one-to-one* if

  each element of B has at most one element of A mapped into it

- f: $A \rightarrow B$ is *onto* B if

  each element of B has at least one element of A mapped into it

(1)To show f is one-to-one,

show that $f(a_1) = f(a_2)$ implies $a_1 = a_2$

(2)To show that f is onto B

show that $\forall b \in B$, $\exists a \in A$ s.t. $f(a) = b$

- *Permutation* of a set A is a function from A into A that is both *one-to-one* and *onto*.

- Terms:

  one-to-one map: *injection*

  onto map: *surjection*

  one-to-one and onto: *bijection*

The Expressive Power of Relational Algebra

- Relational Algebra

$$\sigma, \quad \Pi, \quad \times, \quad \cup, \quad \rho, \quad \div, \quad \bowtie, \quad \bowtie_\theta$$

■ Can we express in R.A. *all meaningful* queries?

➔ Can we obtain all meaningful results using R.A.?

→ Can we characterize the relations, that given a database instance, we can obtain as a result of R.A. expressions?

We will try!

■ For simplicity, consider domain *D* with no order

<, > are meaningless and

consider only =, and ≠ in selections

- R.A. expressions cannot create values!
  - ⇨ A value appears in the result only if it appears in some operand

- Let **r** = {r_1, …, r_n} be a database instance. *Active domain* of **r**, $D_{\mathbf{r}} \subseteq D$, contains values that appear in **r** as values for some tuple of some relation.

- Result of an expression over **r** contains only values from $D_{\mathbf{r}} \cup$ *constant-relations*, if any
  (we will assume no constant relations in expressions)

- The relations that can be generated by means of R.A. expressions involving only relations from **r**
  - ♦ may have any scheme, and
  - ♦ may contain only values that come from $D_{\mathbf{r}}$

■ Example (Figure 2.7)

r_0

Employee	Dept	Secretary
Smith	CS	White
Jones	CS	White
Smith	EE	White
Jones	EE	White
Smith	CS	Grey
Jones	CS	Grey
Smith	EE	Grey
Jones	EE	Grey

r_1

Employee	Dept	Secretary
Smith	CS	White
Jones	CS	Grey

Relation r_1 can only be created value by value; all values are explicitly mentioned in the selection predicate

$$r_1 = \sigma_{(D='CS') \wedge ((E='Smith' \wedge S='White') \vee (E='Jones' \wedge S='Grey'))}(r_0)$$

$$r_2 = \sigma_{(E='Smith')}(r_0)$$

Employee	Dept	Secretary
Smith	CS	White
Smith	EE	White
Smith	CS	Grey
Smith	EE	Grey

The values for each attribute cannot be distinguished from each other by means of the tuple to which they belong

=> related in the same way to the values for the other attributes

Smith ↔ Jones
White ↔ Grey
EE ↔ CS

■ Example (Figure 2.8)

s_0

Predecessor	Successor
William Ⅰ	William Ⅱ
William Ⅱ	Henry Ⅰ
Henry Ⅰ	Stephen

There are relational algebra expressions that can single out the individual values without mentioning them explicitly

$$s_1 = \rho_{\text{Founder} \leftarrow \text{Pred}}(\Pi_{\text{Pred}}(s_0)) - \rho_{\text{Founder} \leftarrow \text{Succ}}(\Pi_{\text{Succ}}(s_0))$$

Founder
William Ⅰ

$$s_2 = \Pi_{\text{Second}}(s_1 \bowtie \rho_{\text{Founder,Second} \leftarrow \text{Pred,Succ}}(s_0))$$

Second
William Ⅱ

Distinguishable Values

(by means of mutual relationships)

- A function $h : D \rightarrow D$ is an *automorphism* of a database instance **r** if
  ① it is a permutation of $D_{\mathbf{r}}$
  ② when we extend its definition to tuples, relations, and database instances, we obtain the identity function on **r**

$$h(\mathbf{r}) = \mathbf{r}$$

i.e., renaming of values in $D_{\mathbf{r}}$ that leaves the database instance invariant

$$t \in r \quad \text{iff} \quad h(t) \in r$$

- Example 2.4:

One automorphism on r_0 of Figure 2.7;

h(Jones) = Smith, h(Smith) = Jones,
h(CS) = EE, h(EE) = CS,
h(White) = White, h(Grey) = Grey

$$h(r_0) = r_0$$

The only automorphism on r_0 of Figure 2.8 is the identity function.

- Two values are *undistinguishable* from each other if the function that
  - (1) exchanges them, and
  - (2) is the identity on other values

  is an automorphism

- A value is *distinguishable* in an instance if all automorphisms of the instance are identity on it.

Automorphisms for selections using constants

Let $C \subseteq D_r$

- *C-fixed automorphism* is an automorphism that is identity on *C*.
- A relational Algebra expression *mentions C* if all constants appearing in the selection predicate belong to *C*.

- $r_1 = \sigma_{(D='CS') \wedge ((E='Smith' \wedge S='White') \vee (E='Jones' \wedge S='Grey'))}(r_0)$

  expression mentions *C*={CS, Smith, White, Jones, Grey}

- {CS, Smith, White, Jones, Grey}-fixed automorphism:

  i (identity)

- {Smith}-fixed automorphism:

  g(Smith) = Smith, g(Jones) = Jones,
  g(CS) = EE, g(EE) = CS,
  g(White) = Grey, g(Grey) = White

Theorem 2.1 Let **r** be a database instance and $C \subseteq D_{\mathbf{r}}$. A relation *r* can be obtained as the result of a relational algebra expression on **r** mentioning *C* iff *r* is invariant by every *C*-fixed automorphism of **r**.

Understanding Theorem 2.1

(1) $C = \emptyset$:

r can be obtained as the result of an expression that does not mention constants iff it is invariant by every automorphism of the db instance.

- relationships between values = *semantic content*
- automorphisms represent uncertainty (indistinguishable)
- *invariant relations* preserve this uncertainty
- others: contain more info then originally contained in the database instance

Example:

h(Jones) = Smith, h(Smith) = Jones,
h(CS) = EE, h(EE) = CS,
h(White) = White, h(Grey) = Grey

$h(r_1)$

Employee	Dept	Secretary
Jones	EE	White
Smith	EE	Grey

$h(r_2)$

Employee	Dept	Secretary
Jones	EE	White
Jones	CS	White
Jones	EE	Grey
Jones	CS	Grey

Both r_1 and r_2 are not invariant under h. Thus, we cannot find an expression that generates these relations without mentioning any constants.

(2) $C \neq Ø$:

fix certain constants (distinguishing between values that are otherwise undistinguishable)

Example: Consider r_2 and g

$r_2 = \sigma_{(E='Smith')}(r_0)$

Employee	Dept	Secretary
Smith	CS	White
Smith	EE	White
Smith	CS	Grey
Smith	EE	Grey

g : {Smith}-fixed automorphism:

g(Smith) = Smith, g(Jones) = Jones,
g(CS) = EE, g(EE) = CS,
g(White) = Grey, g(Grey) = White

Since r_2 is invariant by every {Smith}-fixed automorphism, it can be obtained as the result of a relational algebra expression mentioning {Smith}.

However, r_1 can be obtained by an expression mentioning C = {CS, Smith, White, Jones, Grey}

(3) $C = r$:

r is trivially invariant by any C_r-fixed automorphism

Conclusion:

Relational algebra expressions can construct all relations that contain only information already in the database instance.

cf) Relational Algebra	vs	Datalog
		Relational Calculus
		SQL