Relational Calculus

Databases, Aarhus University

Ira Assent

Intended learning outcomes

- Be able to
  - Read and write basic relational calculus expressions
  - Compare tuple calculus and domain calculus

ira@cs.au.dk

Review: relational algebra

- Relational algebra is basic set of operations for the relational model
  - Formal foundation for relational model operations
    - Express SQL query in relation algebra
    - Basis for implementing and optimizing queries
    - Many concepts incorporated into SQL
- Database relations form an algebra with the operators

  union $\cup$, intersection $\cap$, set difference \ , projection π , renaming ρ, selection σ , Cartesian product $\times$

  plus convenient additional operators natural join *, theta join $\bowtie_{\theta}$, division $\div$
- Combine expressions e.g.

$$\pi_{\text{what,meetid}}(\sigma_{\text{status='acc'}} (\rho_{\text{owner}\to\text{userid}} (\text{Meetings}) * \rho_{\text{participant}\to\text{userid}} (\text{Participants})))$$

ira@cs.au.dk

Additional Relational Operations

- Generalized projection $\pi_{F1,\ F2,\ \ldots,\ Fn}\ (R)$
  - Allows functions of attributes to be included in the projection list
    - Aggregate functions and grouping e.g. `SUM, AVERAGE, MINIMUM`
      - $_{\texttt{course}}\mathcal{F}_{\texttt{COUNT name}}\ (\texttt{Students})$

$$_{<\text{grouping attributes}>}\ \Im\ _{<\text{function list}>}\ (R)$$

- Group tuples by the value of some of their attributes
  - Apply aggregate function $\mathcal{F}$ independently to each group

Students

name	course
Mads	DB
Mads	Prog
Ann	DB
Rie	DB
Rie	Prog
Kurt	Prog

a. $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}(_{\text{Dno}}\ \Im\ _{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE}))$.

b. $_{\text{Dno}}\ \Im\ _{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

c. $\Im\ _{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

(a)

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

(b)

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

(c)

Count_ssn	Average_salary
8	35125

ira@cs.au.dk

Recursive Queries

- Operation applied to **recursive relationship** between tuples of same type
  - Join table with itself
  - Example: find ssns of all employees directly supervised by James Borg

$$\text{BORG\_SSN} \leftarrow \pi_{\text{Ssn}}(\sigma_{\text{Fname}=\text{'James' AND Lname}=\text{'Borg'}}(\text{EMPLOYEE}))$$
$$\text{SUPERVISION}(\text{Ssn1}, \text{Ssn2}) \leftarrow \pi_{\text{Ssn},\text{Super\_ssn}}(\text{EMPLOYEE})$$
$$\text{RESULT1}(\text{Ssn}) \leftarrow \pi_{\text{Ssn1}}(\text{SUPERVISION} \bowtie_{\text{Ssn2}=\text{Ssn}} \text{BORG\_SSN})$$

  - Find Ssns of all employees supervised by someone supervised by James Borg

$$\text{RESULT2(Ssn)} \leftarrow \pi_{\text{Ssn1}} (\text{SUPERVISION} \bowtie_{\text{Ssn2=Ssn}} \text{RESULT1})$$

  - Cannot specify a general query to find Ssns of anyone supervised by James Borg at any level – directly, one level below, two levels below,…

SUPERVISION

(Borg's Ssn is 888665555)

Ssn1 (Ssn)	Ssn2 (Super_ssn)
123456789	333445555
333445555	888665555
999887777	987654321
987654321	888665555
666884444	333445555
453453453	333445555
987987987	987654321
888665555	null

RESULT1

Ssn
333445555
987654321

(Supervised by Borg)

RESULT2

Ssn
123456789
999887777
666884444
453453453
987987987

(Supervised by Borg's subordinates)

Recursive Closure

- Cannot compute the transitive closure of a binary relation R

$$R^{\infty} = \{ (x_1,x_k) \mid \exists x_1,\ldots,x_{k-1} \; ((x_i,x_{i+1}) \in R) \}$$

  - Means that Relational Algebra is not Turing complete
  - But that also means easier to optimize queries
- Another example:
  - Which cities can be reached from Copenhagen in one or more flights?
  - Cannot be specified for arbitrarily many hops
- Arbitrary transitive closure operation has been proposed to compute recursive relationship (also for SQL in SQL3 standard)

from	**to**
Cph	Madrid
Rome	London
Madrid	Athens
Athens	Rome
…	…

ira@cs.au.dk

OUTER JOIN Operators

- **Outer join R ⟗ S**
  - Keep all tuples in *R*, and all those in *S* in both relations regardless of whether or not they have matching tuples in the other relation
    - Similar to SQL
  - **Types**
    - `LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN`
    - If the outer lines only point left, left outer join,
    - if they only point right, right outer join
  - Example left outer join:

$$\text{TEMP} \leftarrow (\text{EMPLOYEE} ⟕_{\text{Ssn=Mgr\_ssn}} \text{DEPARTMENT})$$

$$\text{RESULT} \leftarrow \pi_{\text{Fname, Minit, Lname, Dname}}(\text{TEMP})$$

ira@cs.au.dk

What is equivalent to the following SQL query?

```
SELECT DISTINCT Students.name
    FROM Students, TA
    WHERE Students.name=TA.name;
```

A. $\text{Students} \bowtie_{\text{Students.name=TA.name}} \text{TA}$
B. $\sigma_{\text{Students.name=TA.name}}(\text{Students} \times \text{TA})$
C. $\pi_{\text{Students.name}} (\text{Students} * \text{TA})$
D. $\pi_{\text{name}}(\text{Students}) \cap \pi_{\text{name}} (\text{TA})$

Students

id	name
1	Mads
2	Ann
3	Rie
4	Kurt

TAs

id	name	course	TA class
3	Rie	DB	DA0
4	Kurt	DB	DA99
5	Ann	Prog	DA42
6	Tore	Alg	DA010

Relational Calculus

- Relational algebra describes sequence of operations to derive the desired results
- Relational calculus based on first-order predicate calculus
  - Relational calculus more declarative, specifying what is desired
- Expressive power of the two languages identical
- Many commercial relational query languages based on relational calculus
- Their implementations based on relational algebra
- Two forms of calculi
  - Tuple Relational Calculus (TRC)
  - Domain Relational Calculus (DRC)

ira@cs.au.dk

Tuple Variables and Range Relations

- Tuple Relational Calculus (TRC) expresses results as sets of tuples that satisfy a condition
  - Makes use of **tuple variables** e.g. variable t
    - **Range** over a particular database relation e.g. R
      - Means t is a tuple in R, t ∈ R, written R(t)
- A result of all tuples satisfying some condition COND(*t*)

$$\{t \mid \text{COND}(t)\}$$

- Specify:
  - Range relation *R* of *t*
  - Select particular combinations of tuples
  - Set of attributes to be retrieved (requested attributes)

ira@cs.au.dk

Expressions and Formulas in Tuple Relational Calculus

- General expression of tuple relational calculus is of the form

  $\{t_1.A_j, t_2.A_k, \ldots, t_n.A_m \mid \text{COND}(t_1, t_2, \ldots, t_n, t_{n+1}, t_{n+2}, \ldots, t_{n+m})\}$

  where A_i are attributes in tuple t_j
- **Formula** (Boolean condition)
  - Made up of one or more **atoms** connected via **logical operators** AND, OR, NOT; also written using $\wedge, \vee, \neg$
  - **Atoms** are
    - $R(t_i)$: tuples part of relation R evaluate to TRUE; else FALSE
    - $t_i.A$ op $t_j.B$
    - $t_i.A$ op c or c op $t_j.B$
      - where **comparison operator op** $\in \{=, >, \geq, <, \leq, \neq\}$
      - A attribute of the relation over which t_i ranges, B attribute of the relation over which t_j ranges

  - **Truth value** of an atom
    - Evaluates to either TRUE or FALSE for a specific combination of tuples

ira@cs.au.dk

Tuple Relational Calculus (TRC) examples

- Example:
  - Customer (<u>CustomerID</u>, Name, Street, City, State)
  - Product(<u>ProductID</u>, Name, Price, Category)
  - Purchased(<u>CustomerID</u>, <u>ProductID</u>, <u>Date</u>)
- List all information about expensive products (here conveniently defined as costing more than € 1000) $\{t \mid Product(t) \wedge t.name = \text{'Cookie'}\}$
  - *Product(t)* specifies the range relation *Product* for the tuple variable *t*
  - Each tuple satisfying *t.name* = *'Cookie'* is retrieved
  - The entire tuple is retrieved

ProductID	Name	Price	Category
1	Pule Cheese	4000	Dairy
2	Cookie	15	Sweets
3	Rolex	99000	Jewelry

- TRC example: List the names of dairy products costing more than 1000

ira@cs.au.dk

Logical Operators and Quantifiers

- Combining conditions using logical operators
  - List the extreme price products (over €1000 or under €1)

    $\{t \mid Product(t) \wedge (t.Price > 1000 \vee t.Price < 1)\}$

Customer (<u>CustomerID</u>, Name, Street, City, State)
Product(<u>ProductID</u>, Name, Price, Category)
Purchased(<u>CustomerID</u>, <u>ProductID</u>, <u>Date</u>)

- We can use quantifiers $\forall$, $\exists$ from predicate calculus with tuple variables
    - Universal quantifier $\forall$ true if true for every tuple
    - Existential quantifier $\exists$ true if true for any tuple
  - List the products where there is at least one purchased item from the product's category

    $\{t \mid Product(t) \wedge \exists s\ (Purchased(s)\ t.Category = s.Category)\}$
  - A tuple variable is **bound** if it is quantified, otherwise **free**

ira@cs.au.dk

Queries

- List the names of customers who have purchased a product

- List the IDs of expensive (priced at more than 1000) purchased products

Customer (CustomerID, Name, Street, City, State)

Product(ProductID, Name, Price, Category)

Purchased(CustomerID, ProductID, Date)

ira@cs.au.dk

Nested queries

- List the customers who have purchased all soy products

Customer (<u>CustomerID</u>, Name, Street, City, State)

Product(<u>ProductID</u>, Name, Price, Category)

Purchased(<u>CustomerID</u>, <u>ProductID</u>, <u>Date</u>)

$$\{c.Name \mid Customer(c) \wedge \forall f\,(Product(f) \wedge f.Category = \text{“Soy”} \Rightarrow (\exists r\,(Purchased(r) \wedge r.ProductID = f.ProductID \wedge r.CustomerID = c.CustomerID)))\}$$

- if a product is in the soy category, then it is purchased
- a $\Rightarrow$ b is shorthand for $\neg$a $\vee$ b.

ira@cs.au.dk

Transforming expressions

- Transform one type of quantifier into other with negation (preceded by NOT)
  - $\forall x\ (Cat(x)) \equiv \neg\ \exists x\ (\neg\ Cat(x))$
  - All x are cats means there is no x that is not a cat
- AND and OR replace one another
  - **Query 3.** List the names of employees who work on *all* the projects controlled by department number 5. One way to specify this query is to use the universal
  - quantifier as shown:

Q3: {*e*.Lname, *e*.Fname | EMPLOYEE(*e*) **AND** ((∀*x*)(**NOT**(PROJECT(*x*)) **OR NOT** (*x*.Dnum=5) **OR** ((∃ *w*)(WORKS_ON(*w*) **AND** *w*.Essn=*e*.Ssn **AND** *x*.Pnumber=*w*.Pno))))}

Q3A: {*e*.Lname, *e*.Fname | EMPLOYEE(*e*) **AND** (**NOT** (∃*x*) (PROJECT(*x*) **AND** (*x*.Dnum=5) AND (**NOT** (∃ *w*)(WORKS_ON(*w*) **AND** *w*.Essn=*e*.Ssn **AND** *x*.Pnumber=*w*.Pno))))}

ira@cs.au.dk

Equivalences summarized

- $\forall x\,(P(x)) \equiv \neg \exists x\,(\neg P(x))$
- $\exists x\,(P(x)) \equiv \neg \forall x\,(\neg P(x))$
- $\forall x\,(P(x) \wedge Q(x)) \equiv \neg \exists x\,(\neg P(x) \vee \neg Q(x))$
- $\forall x\,(P(x) \vee Q(x)) \equiv \neg \exists x\,(\neg P(x) \wedge \neg Q(x))$
- $\exists x\,(P(x) \vee Q(x)) \equiv \neg \forall x\,(\neg P(x) \wedge \neg Q(x))$
- $\exists x\,(P(x) \wedge Q(x)) \equiv \neg \forall x\,(\neg P(x) \vee \neg Q(x))$
- $\forall x\,(P(x)) \Rightarrow \exists x\,(P(x))$
- $\neg \exists x\,(P(x)) \Rightarrow \neg \forall x\,(P(x))$

ira@cs.au.dk

Finding the maximum

- Find the product(s) with maximum price – without using a maximum (or minimum) operator!

Customer (<u>CustomerID</u>, Name, Street, City, State)

Product(<u>ProductID</u>, Name, Price, Category)

Purchased(<u>CustomerID</u>, <u>ProductID</u>, <u>Date</u>)

ira@cs.au.dk

Equivalence

- What does this query say?

$$\{c.Street \mid \exists r\ \exists f\ (Customer(c) \wedge Purchased(r) \wedge Product(f) \wedge f.Category = \text{“Dairy”} \wedge c.CustomerID = r.CustomerID \wedge r.ProductID = f.ProductID)\}$$

Customer (<u>CustomerID</u>, Name, Street, City, State)

Product(<u>ProductID</u>, Name, Price, Category)

Purchased(<u>CustomerID</u>, <u>ProductID</u>, <u>Date</u>)

A. SELECT street FROM Customer, Product, Purchased WHERE Category ='Dairy';

B. SELECT street FROM Customer WHERE Customer. CustomerID=Purchased.CustomerID AND Purchased.ProductID=Product.ProductID AND Category='Dairy';

C. SELECT c FROM Customer WHERE Customer. CustomerID=Purchased.CustomerID AND Purchased.ProductID=Product.ProductID AND Category='Dairy';

D. SELECT street FROM Customer, Product, Purchased WHERE Customer. CustomerID=Purchased.CustomerID AND Purchased.ProductID=Product.ProductID AND Category='Dairy';

ira@cs.au.dk

Safety

- possible to write tuple calculus expression that generates infinite relations
- $\{t \mid \neg\, r(t)\}$ results in infinite relation if the domain of any attribute of relation *r* is infinite
  - E.g. $\{t \mid \neg\, Employee(t)\}$
    - All tuples which are not employees: infinitely many!
  - **domain** of tuple relational calculus expression is set of all values that either appear as
    - constant values in the expression or that
    - exist in any tuple of the relations referenced in the expression
- ensure that an expression in relational calculus yields only finite number of tuples
  - An expression is **safe** if all values in its result are from the domain of the expression
    - Do not want to consider infinite set of values
    - Means: do not write such expressions!

Safe?

- $\{ t \mid t.A = 5 \vee true \}$
- $\{ t \mid t.A = 5 \vee t.B = t.B \}$

1. Yes
2. Only upper one
3. Only lower one
4. No

Safe?

- $\{ t \mid t.A = 5 \vee \text{true} \}$
  - Not safe, because infinite tuples possible
    - "true" not limited to any tuples
- $\{ t \mid t.A = 5 \vee t.B = t.B \}$
  - Safe, because limited to tuples from the domain
    - We can check all tuples if they fulfil either condition
- $\{t \mid \exists\, r\ Student(r) \wedge (t.ID = r.ID) \wedge (\forall\, u\ Course(u)\ (u.dept\_name = \text{"CS"} \Rightarrow \exists\, s\ Takes(s) \wedge (t.ID = s.ID \wedge s.course\_id = u.course\_id))\}$
  - Without existential quantifier on Student: not safe if there is no course offered by the CS department
    - $\{t \mid \forall\, u\ Course(u)\ (u.dept\_name = \text{"CS"} \Rightarrow \exists\, s\ Takes(s) \wedge (t.ID = s.ID \wedge s.course\_id = u.course\_id)\}$ unsafe
    - because would then be infinitely many possible tuples to consider
    - So, make sure to limit to domain!

Domain Relational Calculus

- Domain calculus differs from tuple calculus in the type of variables used in formulas
  - Rather than variables ranging over tuples, **ranges over single values** from domains of attributes
  - To form a relation of degree n, need n domain variables
  - Otherwise, as in tuple relational calculus
- Each query is an expression of the form

  $$\{ x_1, x_2, \dots, x_n \mid P(x_1, x_2, \dots, x_n) \}$$

  - P is a formula of the domain calculus (that is, a condition)
- List all information on expensive products

  $$\{ I, N, P, C \mid Product(I, N, P, C) \wedge P > 1000 \}$$

- List the names of the expensive products

  $$\{ N \mid \exists I\, \exists P\, \exists C\, (Product(I, N, P, C) \wedge P > 1000) \}$$

Customer (<u>CustomerID</u>, Name, Street, City, State)

Product(<u>ProductID</u>, Name, Price, Category)

Purchased(<u>CustomerID</u>, <u>ProductID</u>, <u>Date</u>)

ira@cs.au.dk

Query example

- List the IDs of expensive products that have not been purchased

Customer (<u>CustomerID</u>, Name, Street, City, State)

Product(<u>ProductID</u>, Name, Price, Category)

Purchased(<u>CustomerID</u>, <u>ProductID</u>, <u>Date</u>)

ira@cs.au.dk

Domain Relational Calculus (DRC) textbook example

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

Q1: $\{q, s, v \mid (\exists z)\ (\exists l)\ (\exists m)\ (\text{EMPLOYEE}(qrstuvwxyz)\ \textbf{AND}\ \text{DEPARTMENT}(lmno)\ \textbf{AND}\ l=\text{'Research'}\ \textbf{AND}\ m=z)\}$

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, birth date, and address.

Q2: $\{i, k, s, u, v \mid (\exists j)(\exists m)(\exists n)(\exists t)(\text{PROJECT}(hijk)\ \textbf{AND}\ \text{EMPLOYEE}(qrstuvwxyz)\ \textbf{AND}\ \text{DEPARTMENT}(lmno)\ \textbf{AND}\ k=m\ \textbf{AND}\ n=t\ \textbf{AND}\ j=\text{'Stafford'})\}$

- Note: QBE language for relational databases based on domain relational calculus (IBM)

ira@cs.au.dk

List the customers who have purchased expensive products

A. $\{K \mid Customer(K, A, S, B, L)$
$\quad \wedge \exists I,D\ (Purchased(K, I, D)$
$\quad\quad \wedge (\exists N,P,C\ (Product(I, N, P, C) \wedge P > 1000)))\}$

B. $\{K \mid \exists A,S,B,L\ (Customer(K, A, S, B, L)$
$\quad \wedge \exists I,D\ (Purchased(K, I, D)$
$\quad\quad \wedge (\exists N,P,C\ (Product(I, N, P, C) \wedge P > 1000)))\}$

C. $\{K \mid \exists A,S,B,L\ (Customer(K, A, S, B, L)$
$\quad \wedge \exists K,I,D\ (Purchased(K, I, D)$
$\quad\quad \wedge (\exists N,P,C\ (Product(I, N, P, C) \wedge P > 1000)))\}$

D. $\{K \mid \exists K,A,S,B,L\ (Customer(K, A, S, B, L)$
$\quad \wedge \exists M,I,D\ (Purchased(M, I, D)$
$\quad\quad \wedge (\exists I,N,P,C\ (Product(I, N, P, C) \wedge P > 1000)))\}$

Customer (<u>CustomerID</u>, Name, Street, City, State)

Product(<u>ProductID</u>, Name, Price, Category)

Purchased(<u>CustomerID</u>, <u>ProductID</u>, <u>Date</u>)

ira@cs.au.dk

Assertions

- Assertions: general integrity constraints
  - expressed directly as predicates which must always be satisfied
  - in the algebra or calculi, of the form
    - There does not exist an offending tuple

- Example: No product has negative price
  - Algebra
    - $\sigma_{Price < 0}(Product) = \{\}$
  - Tuple Relational Calculus
    - $\neg \exists f(Product(f) \wedge f.Price < 0)$
  - Domain Relational Calculus
    - $\neg \exists I,N,P,C(Product(I,N,P,C) \wedge P < 0)$

Customer (<u>CustomerID</u>, Name, Street, City, State)

Product(<u>ProductID</u>, Name, Price, Category)

Purchased(<u>CustomerID</u>, <u>ProductID</u>, <u>Date</u>)

ira@cs.au.dk

Expressive Power

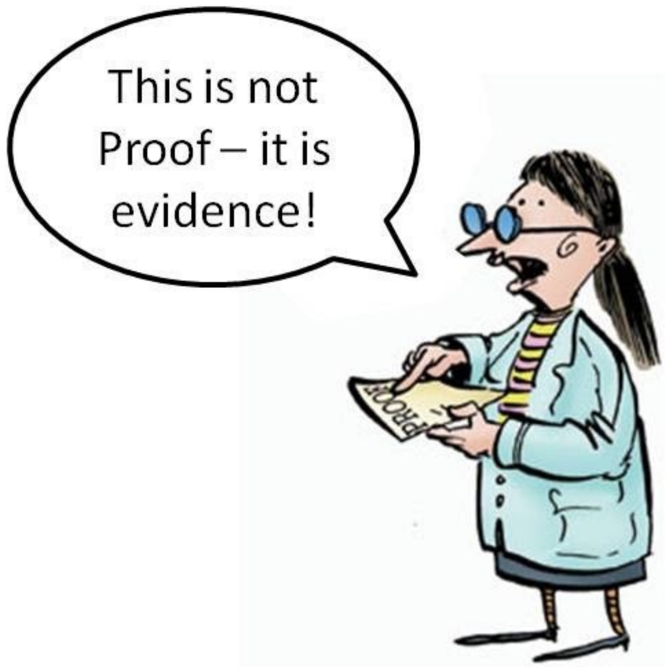
- The following three languages define the same class of functions
  - Relational algebra expressions
  - Safe relational tuple calculus formulas
  - Safe relational domain calculus formulas
- Corollary: All three languages are relationally complete

ira@cs.au.dk

Equivalence of Expressive Power

- Theorem: The relational algebra is as expressive as the (safe) tuple relational calculus

- Proof idea: by induction on the number of operators in the calculus predicate
  - $\neg P(r) \Rightarrow U \setminus r$
  - $P(r) \Rightarrow \sigma_P(\ldots)$
  - $X \wedge Y \Rightarrow \neg(\neg X \vee \neg Y)$
  - $\forall X(P(r)) \Rightarrow \neg \exists X(\neg P(r))$
  - $\exists X(\ldots) \Rightarrow \pi(\ldots)$
  - $X \vee Y \Rightarrow \pi(X \times Y)$



Summary

- Intended learning outcomes
  - Be able to
    - Read and write basic relational calculus expressions
    - Compare tuple calculus and domain calculus

- Acknowledgements
  - Richard T. Snodgrass (University of Arizona), Christian S. Jensen (Aalborg University), Kristian Torp (Aalborg University), Curtis Dyreson (Washington State University)

ira@cs.au.dk

What was this all about?

Guidelines for your own review of today's session

- Tuple relational calculus is…
  - Variables range over …
  - Each variable is associated with …
  - The basic form of a query is…
  - Safe queries are…
- Domain relational calculus…
  - Variables range over…
  - Each variable is associated with…
  - The basic form of a query is…
- Assertions are expressed as…
- The expressive power…

ira@cs.au.dk