Chapter 4

Relational Algebra

&

Relational Calculus

Chapter 4 - Objectives

- **Meaning of the term relational completeness.**
- **How to form queries in relational algebra.**
- **How to form queries in tuple relational calculus.**
- **How to form queries in domain relational calculus.**
- **Categories of relational DML.**

Introduction

- **Relational algebra and relational calculus are formal languages associated with the relational model.**
- **Informally, relational algebra is a (high-level) procedural language and relational calculus a non-procedural language.**
- **However, formally both are equivalent to one another.**
- **A language that produces a relation that can be derived using relational calculus is relationally complete.**

Database Query Languages

- Given a database, ask questions, get data as answers
  - Get all students with GPA > 3.7 who applied to USQ and QUT and nowhere else
  - Get all humanities departments at campuses in Queensland with < 200 applicants
  - Get the campus with highest average accept rate over the last five years
- Some questions are easy to pose, some are not
- Some questions are easy for DBMS to answer, some are not.
- "Query language“, but also used to update the database

Relational Query Languages

- Formal:
  - relational algebra, relational calculus, Datalog
- Practical:
  - SQL,
  - Quel,
  - Query-by-Example (QBE)
- In **ALL** languages, a query is executed over a set of relations, get single relation as the result

Relational Algebra (RA)

- **Relational algebra operations work on one or more relations to define another relation without changing the original relations.**

- **Both operands and results are relations, so output from one operation can become input to another operation.**

- **Allows expressions to be nested, just as in arithmetic. This property is called closure.**

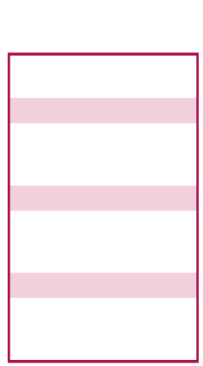
Relational Algebra

- **5 basic operations in relational algebra: Selection, Projection, Cartesian product, Union, and Set Difference.**
- **These perform most of the data retrieval operations needed.**
- **Also have Join, Intersection, and Division operations, which can be expressed in terms of 5 basic operations.**

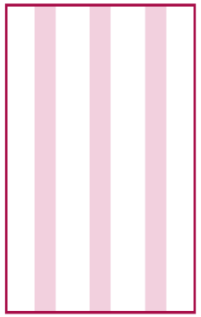
RA Operations

- Operations of traditional relational algebra fall into four broad classes:
  1. Operations that remove parts of a relation
  2. Renaming
  3. Set operations
  4. Operations that combine tuples of two relations

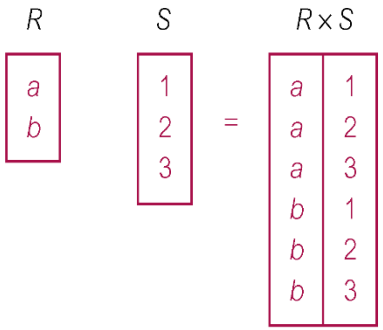
Relational Algebra Operations



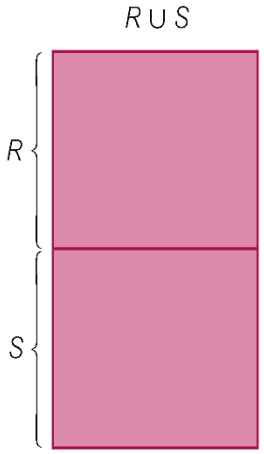
(a) Selection



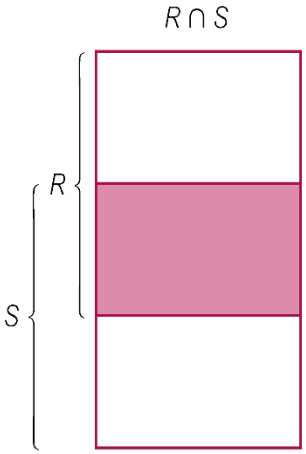
(b) Projection



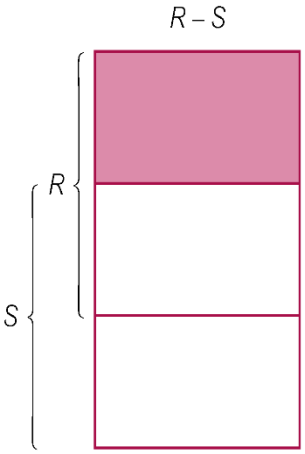
(c) Cartesian product



(d) Union

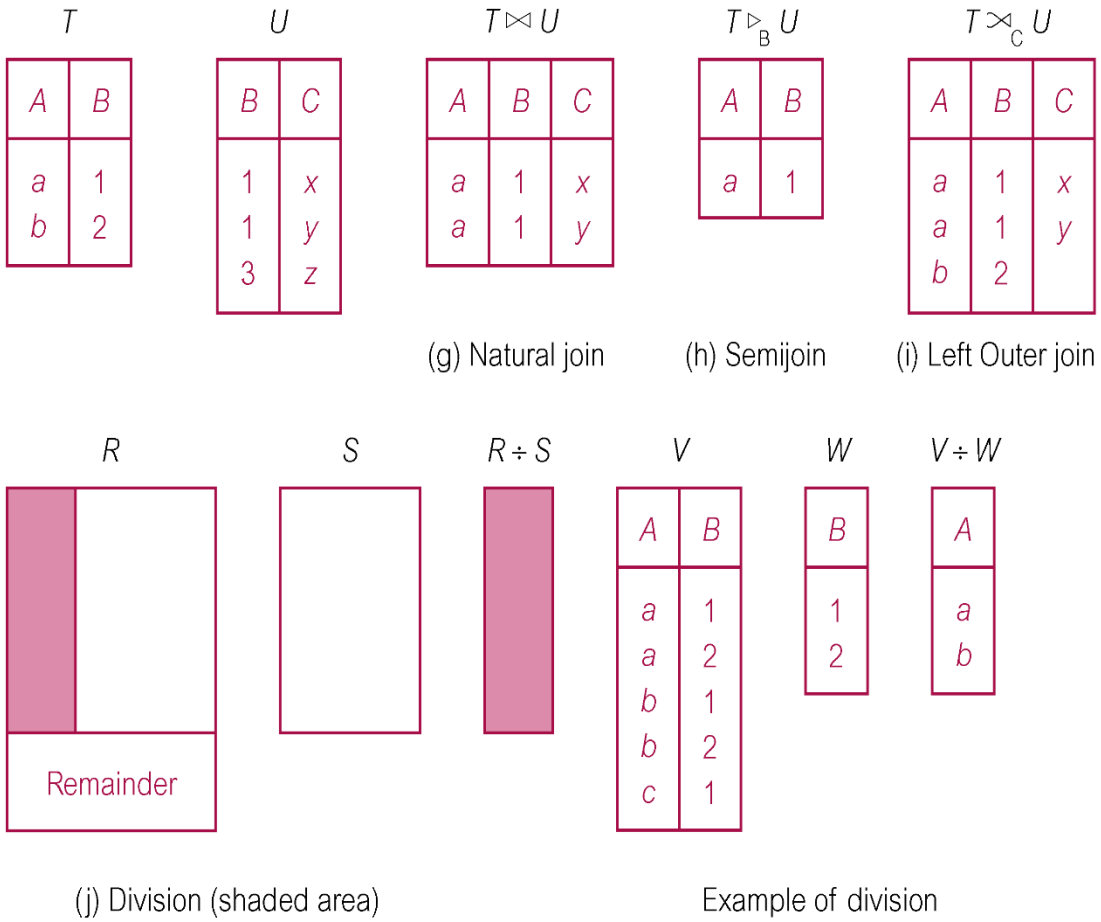


(e) Intersection



(f) Set difference

Relational Algebra Operations



Selection (or Restriction)

- $\sigma_{\text{predicate}}$ **(*R*)**
  - **Works on a single relation *R* (unary operation) and defines a relation that contains only those tuples (rows) of *R* that satisfy the specified condition (*predicate*).**
  - Schema of σ_C (*R*) is the same as schema of R
  - Selection loses information
  - Condition *C*:
    - AND, OR, NOT, A θ B, A θ c, where θ {<, ≤, >, ≥, =, ≠}

Example - Selection (or Restriction)

- **List all staff with a salary greater than £10,000.**

$\sigma_{salary > 10000}$ **(Staff)**

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24- Mar-58	18000	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003

Projection

- $\Pi_{col1, \ldots, coln}(R)$
  - **Works on a single relation *R* (unary operation) and defines a relation that contains a vertical subset of *R*, extracting the values of specified attributes and eliminating duplicates.**
  - Projection loses information
    - Possibly vertically, possibly horizontally
  - Schema of resulting relation: attributes subset of the attributes of R

Example - Projection

- **Produce a list of salaries for all staff, showing only staffNo, fName, lName, and salary details.**

$\Pi_{staffNo, fName, lName, salary}(Staff)$

staffNo	fName	lName	salary
SL21	John	White	30000
SG37	Ann	Beech	12000
SG14	David	Ford	18000
SA9	Mary	Howe	9000
SG5	Susan	Brand	24000
SL41	Julie	Lee	9000

Rename

- $\rho_{\gamma}(R)$ (unary operation)
  - γ is a one-to-one function that maps a set of attributes to a new set of attributes
  - Schema is the same, up to renaming of attributes
  - Content, or instance, remains unchanged

Union

- ***R*** ∪ ***S*** (binary operation, set operation)
  - **Union of two relations *R* and *S* defines a relation that contains all the tuples of *R*, or *S*, or both *R* and *S*, duplicate tuples being eliminated.**
  - ***R* and *S* must be union-compatible.**
    - **Same set of attributes + domains**
- **If *R* and *S* have *I* and *J* tuples, respectively, union is obtained by concatenating them into one relation with a *maximum* of (*I* + *J*) tuples.**
- Lossless, but impossible to undo
- commutative, associative

Example - Union

- **List all cities where there is either a branch office or a property for rent.**

Π_{city}(Branch) $\cup$ Π_{city}(PropertyForRent)

city
London
Aberdeen
Glasgow
Bristol

Set Difference

- ***R* – *S*** (binary operation, set operation)
  - **Defines a relation consisting of the tuples that are in relation *R*, but not in *S*.**
  - ***R* and *S* must be union-compatible.**
  - Loses information
  - $R - S \neq S - R$!!

Example - Set Difference

- **List all cities where there is a branch office but no properties for rent.**

Π_{city}(Branch) – Π_{city}(PropertyForRent)

city
Bristol

Intersection

- ***R*** ∩ ***S*** (binary operation, set operation)
  - **Defines a relation consisting of the set of all tuples that are in both *R* and *S*.**
  - ***R* and *S* must be union-compatible.**

- **Expressed using basic operations:**

  $R \cap S = R - (R - S)$

- commutative, associative

Example - Intersection

- **List all cities where there is both a branch office and at least one property for rent.**

Π_{city}(Branch) $\cap$ Π_{city}(PropertyForRent)

city
Aberdeen
London
Glasgow

Cartesian product

- ***R* × *S* (binary operation, set operation)**
  - **Defines a relation that is the concatenation of every tuple of relation *R* with every tuple of relation *S*.**
  - Lossless, possible to undo using projection
    - Unless one of *R*, *S* is empty!
  - #(*R* x *S*) = #*R* * #*S*
  - Schema: union of sets of attributes
  - commutative, associative

Example - Cartesian Product

- List the names and comments of all clients who have viewed a property for rent.

  $(\Pi_{clientNo, fName, lName}(Client)) \times (\Pi_{clientNo, propertyNo, comment}(Viewing))$

  Requires further restriction!

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR56	PA14	too small
CR76	John	Kay	CR76	PG4	too remote
CR76	John	Kay	CR56	PG4	
CR76	John	Kay	CR62	PA14	no dining room
CR76	John	Kay	CR56	PG36	
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR62	PA14	no dining room
CR56	Aline	Stewart	CR56	PG36	
CR74	Mike	Ritchie	CR56	PA14	too small
CR74	Mike	Ritchie	CR76	PG4	too remote
CR74	Mike	Ritchie	CR56	PG4	
CR74	Mike	Ritchie	CR62	PA14	no dining room
CR74	Mike	Ritchie	CR56	PG36	
CR62	Mary	Tregear	CR56	PA14	too small
CR62	Mary	Tregear	CR76	PG4	too remote
CR62	Mary	Tregear	CR56	PG4	
CR62	Mary	Tregear	CR62	PA14	no dining room
CR62	Mary	Tregear	CR56	PG36	

Example - Cartesian Product and Selection

- Use selection operation to extract those tuples where Client.clientNo = Viewing.clientNo.

$$\sigma_{\text{Client.clientNo = viewing.clientNo}}((\prod_{\text{clientNo,fName,lName}}(\text{Client})) \text{ X } (\prod_{\text{clientNo,propertyNo,comment}}(\text{Viewing})))$$

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR56	PG36	
CR62	Mary	Tregear	CR62	PA14	no dining room

- Cartesian product and Selection can be reduced to a single operation called a *Join*.

Join Operations

- **Join is a derivative of Cartesian product.**

- **Equivalent to performing a Selection, using join predicate as selection formula, over Cartesian product of the two operand relations.**
  - $\sigma_C (R \times S)$

- **One of the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMSs have intrinsic performance problems.**
  - But can usually be optimized

Join Operations

- Various forms of join operation
  - Theta join
  - Equijoin (a particular type of Theta join)
  - Natural join
  - Outer join
  - Semijoin

Theta join (θ-join)

- $R \bowtie_F S$
  - Defines a relation that contains tuples satisfying the predicate F from the Cartesian product of *R* and *S*.
  - The predicate F is of the form $R.a_i \; \theta \; S.b_i$ where θ may be one of the comparison operators (<, ≤, >, ≥, =, ≠).

Theta join (θ-join)

- Can rewrite Theta join using basic Selection and Cartesian product operations.

$$R \underset{F}{\bowtie} S = \sigma_F(R \times S)$$

- **Degree of a Theta join is sum of degrees of the operand relations *R* and *S*.**
- **If predicate F contains only equality (=), the term *Equijoin* is used.**

Example - Equijoin

- **List the names and comments of all clients who have viewed a property for rent.**

$$(\Pi_{clientNo,fName,lName}(Client)) \quad {}_{Client.clientNo = Viewing.clientNo} \bowtie \; (\Pi_{clientNo,propertyNo,comment}(Viewing))$$

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR56	PG36	
CR62	Mary	Tregear	CR62	PA14	no dining room

Natural Join

- **R ⋈ S**
  - **An Equijoin of the two relations *R* and *S* over all common attributes *x*. One occurrence of each common attribute is eliminated from the result.**
  - Usual simulation (selection and cartesian product), plus projection

Example - Natural Join

- **List the names and comments of all clients who have viewed a property for rent.**

$(\Pi_{clientNo,fName,lName}(Client))$ $(\Pi_{clientNo,propertyNo,comment}(Viewing))$

$\bowtie$

clientNo	fName	lName	propertyNo	comment
CR76	John	Kay	PG4	too remote
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR56	Aline	Stewart	PG36	
CR62	Mary	Tregear	PA14	no dining room

Outer join

- **To display rows in the result that do not have matching values in the join column, use Outer join.**

- **R ⟕ S**
  - **(Left) outer join is join in which tuples from R that do not have matching values in common columns of S are also included in result relation.**
  - Padded with `NULL`s

Example - Left Outer join

- **Produce a status report on property viewings.**

$\Pi_{propertyNo,street,city}$(PropertyForRent) ⟕ Viewing

propertyNo	street	city	clientNo	viewDate	comment
PA14	16 Holhead	Aberdeen	CR56	24-May-01	too small
PA14	16 Holhead	Aberdeen	CR62	14-May-01	no dining room
PL94	6 Argyll St	London	**null**	**null**	**null**
PG4	6 Lawrence St	Glasgow	CR76	20-Apr-01	too remote
PG4	6 Lawrence St	Glasgow	CR56	26-May-01	
PG36	2 Manor Rd	Glasgow	CR56	28-Apr-01	
PG21	18 Dale Rd	Glasgow	**null**	**null**	**null**
PG16	5 Novar Dr	Glasgow	**null**	**null**	**null**

Semijoin

- R $\ltimes_F$ S
  - Defines a relation that contains the tuples of *R* that participate in the join of *R* with *S*.

- **Can rewrite Semijoin using Projection and Join:**

$$R \triangleright_F S = \Pi_A(R \bowtie_F S)$$

Example - Semijoin

- **List complete details of all staff who work at the branch in Glasgow.**

Staff $\triangleright_{\text{Staff.brancNo = Branch.branchNo and branch.city = 'Glasgow'}}$ **Branch**

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24- Mar-58	18000	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003

Division

- $R \div S$
  - Defines a relation over the attributes C that consists of set of tuples from R that match combination of *every* tuple in S.

- Expressed using basic operations:

$$T_1 \leftarrow \Pi_C(R)$$

$$T_2 \leftarrow \Pi_C((S \times T_1) - R)$$

$$T \leftarrow T_1 - T_2$$

Example - Division

- **Identify all clients who have viewed all properties with three rooms.**

$$(\Pi_{clientNo,propertyNo}(Viewing)) \div (\Pi_{propertyNo}(\sigma_{rooms = 3}(PropertyForRent)))$$

$\Pi_{clientNo,propertyNo}(Viewing)$

clientNo	propertyNo
CR56	PA14
CR76	PG4
CR56	PG4
CR62	PA14
CR56	PG36

$\Pi_{propertyNo}(\sigma_{rooms=3}(PropertyForRent))$

propertyNo
PG4
PG36

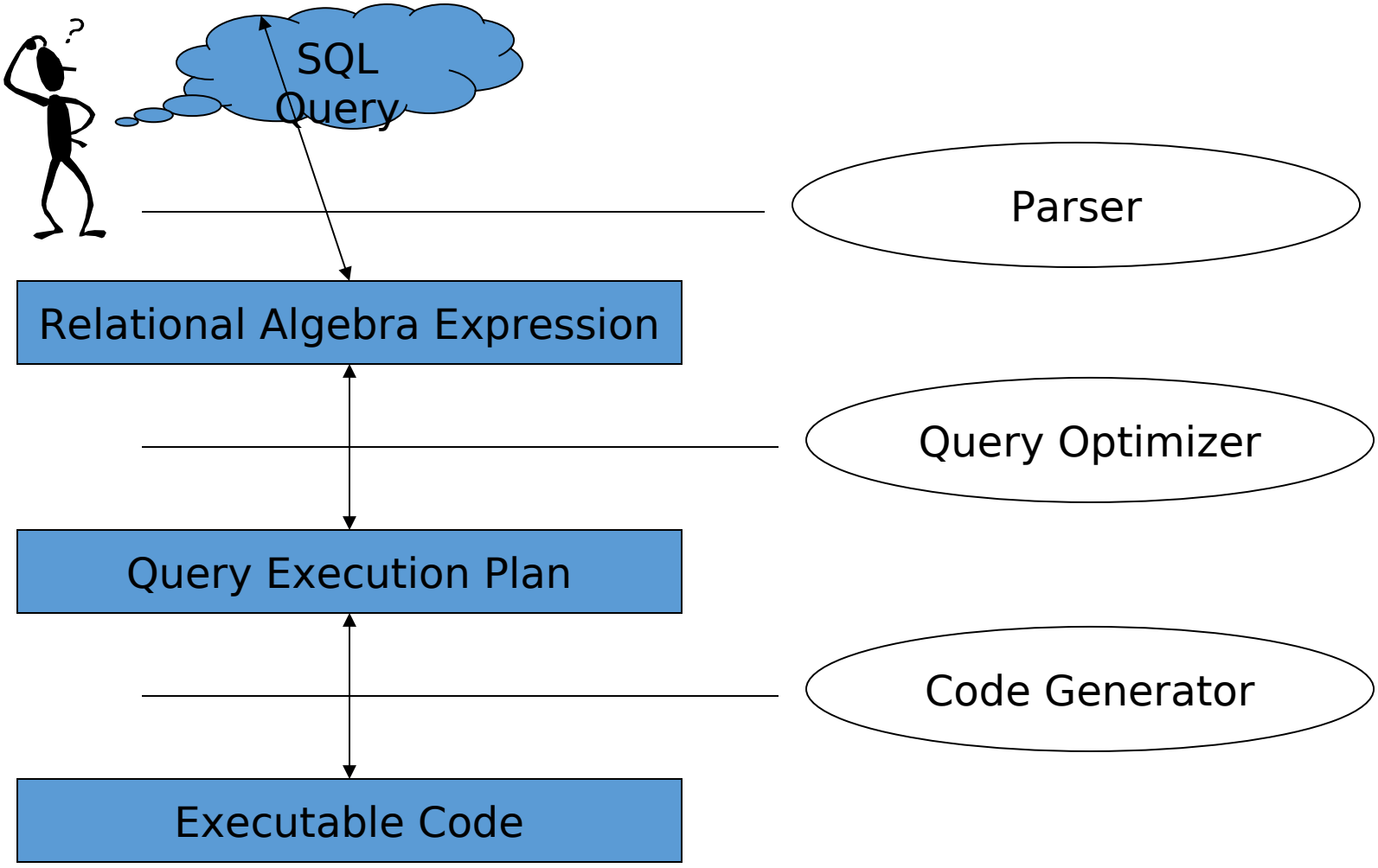
RESULT

clientNo
CR56

Why Relational Algebra?

- All DBMSs use relational algebra as intermediate language for specifying query evaluation algorithms
  - Parse SQL and translate it into expression in relational algebra
  - However, translated expression (or straight SQL) would be very inefficient
- Set of rules for manipulating algebraic expressions
  - Don’t exist for SQL
  - Expressions can be converted into equivalent ones which take less time to execute
  - Done by query optimizer

Overview of Query Processing



Remarks about the Relational Algebra

- The Relational Algebra is **not** Turing Complete
  - No explicit loop
  - No recursion
- This is a feature, not a bug
  - Helps with query optimization and processing
  - Operations are linear in size of instance

} No transitive closure!

- It is undecidable whether two algebra expressions are equivalent
  - Restriction to Conjunctive Queries: decidable
    - CQ: Selection, projection, Cartesian product only

Non-trivial example queries

- Consider the relation schema:
  - Visits(Drinker,Bar); Likes(Drinker,Beer); Serves(Bar,Beer);
- Give all the drinkers with the beers they do not like
  - $(\Pi_{Drinker}(Likes) \times \Pi_{Beer}(Likes)) - Likes$

- Give the pairs of beers that are not served in a common bar
  - $(\Pi_{Beer}(Serves) \times \Pi_{Beer}(Serves)) -$
  - $\Pi_{Beer1,Beer2}.\sigma_{Bar1=Bar2}(Serves \times Serves)$

More hard RA expressions

- Give all the drinkers that like all beers that ‘John’ likes
  - Likes : Π_{Beer} . $\sigma_{Drinker='John'}$ (Likes)
- Give all the drinkers that like exactly the same beers as ‘John’
  - (Likes : Π_{Beer} . $\sigma_{Drinker='John'}$ (Likes)) ∩
  - (($\Pi_{Drinker}$(Likes) X Π_{Beer}(Likes)) - Likes) :
  - Π_{Beer}. $\sigma_{Drinker='John'}$ (($\Pi_{Drinker}$(Likes) X Π_{Beer}(Likes)) - Likes))

Relational Calculus (RC)

- **Relational calculus query specifies *what* is to be retrieved rather than *how* to retrieve it.**
  - **No description of how to evaluate a query.**

- **In first-order logic (or predicate calculus), *predicate* is a truth-valued function with arguments.**

- **When we substitute values for the arguments, function yields an expression, called a *proposition*, which can be either true or false.**

Relational Calculus

- **If predicate contains a variable (e.g. '*x* is a member of staff'), there must be a range for *x*.**
- **When we substitute some values of this range for *x*, proposition may be true; for other values, it may be false.**
- **When applied to databases, relational calculus has two forms: *tuple* and *domain*.**

Tuple Relational Calculus (TRC)

- **Interested in finding tuples for which a predicate is true. Based on use of tuple variables.**

- **Tuple variable is a variable that 'ranges over' a named relation: i.e., variable whose only permitted values are tuples of the relation.**

- **Specify range of a tuple variable *S* as the Staff relation as:**

  Staff(S)

- **To find set of all tuples S such that P(S) is true:**

  {S | P(S)}

Tuple Relational Calculus - Example

- **To find details of all staff earning more than £10,000:**

  {S | Staff(S) ^ S.salary > 10000}

- **To find a particular attribute, such as salary, write**:

  {S.salary | Staff(S) ^ S.salary > 10000}

Tuple Relational Calculus

- **Can use two *quantifiers* to tell how many instances the predicate applies to:**
  - **Existential quantifier ∃ (‘there exists’)**
  - **Universal quantifier ∀ (‘for all’)**

- **Tuple variables qualified by ∀ or ∃ are called *bound* variables, otherwise called *free* variables.**

Tuple Relational Calculus

- **Existential quantifier used in formulae that must be true for at least one instance, such as:**

  Staff(S) ^ (∃B)(Branch(B) ^ (B.branchNo = S.branchNo) ^ B.city = 'London')

- **Means 'There exists a Branch tuple that has the same branchNo as the branchNo of the current Staff tuple, *S*, and is located in London'.**

Tuple Relational Calculus

- **Universal quantifier is used in statements about every instance, such as:**

  (∀B) (B.city ≠ ‘Paris’)

- **Means ‘For all Branch tuples, the address is not in Paris’.**

- **Can also use ~(∃B) (B.city = ‘Paris’) which means ‘There are no branches with an address in Paris’.**

Tuple Relational Calculus

- **Formulae should be unambiguous and make sense.**
- **A (well-formed) formula is made out of <u>atoms</u>:**
  - $R(S_i)$**, where** S_i **is a tuple variable and** R **is a relation**
  - $S_i.a_1 \ \theta \ S_j.a_2$
  - $S_i.a_1 \ \theta \ c$
- **Can recursively build up formulae from atoms:**
  - **An atom is a formula**
  - **If** F_1 **and** F_2 **are formulae, so are their conjunction,** $F_1 \wedge F_2$**; disjunction,** $F_1 \vee F_2$**; and negation,** $\sim F_1$
  - **If** F **is a formula with free variable** X**, then** $(\exists X)(F)$ **and** $(\forall X)(F)$ **are also formulae.**

Example - Tuple Relational Calculus

a) List the names of all managers who earn more than £25,000.

$$\{S.fName, S.lName \mid Staff(S) \wedge S.position = \text{'Manager'} \wedge S.salary > 25000\}$$

b) List the staff who manage properties for rent in Glasgow.

$$\{S \mid Staff(S) \wedge (\exists P)\,(PropertyForRent(P) \wedge (P.staffNo = S.staffNo) \wedge P.city = \text{'Glasgow'})\}$$

Example - Tuple Relational Calculus

c) List the names of staff who currently do not manage any properties.

$$\{S.fName, S.lName \mid Staff(S) \wedge (\neg(\exists P)\ (PropertyForRent(P) \wedge (S.staffNo = P.staffNo)))\}$$

Or

$$\{S.fName, S.lName \mid Staff(S) \wedge (\forall P)\ (\neg PropertyForRent(P) \vee \neg(S.staffNo = P.staffNo)))\}$$

Example - Tuple Relational Calculus

- **List the names of clients who have viewed a property for rent in Glasgow.**

{C.fName, C.lName | Client(C) ^ ((∃V)(∃P)

(Viewing(V) ^ PropertyForRent(P) ^ (

C.clientNo = V.clientNo) ^

(V.propertyNo=P.propertyNo)^P.city =‘Glasgow’))}

Tuple Relational Calculus

- **Expressions can generate an infinite set. For example:**
  {S | ~Staff(S)}

- **To avoid this** [an *unsafe* query]**, add restriction that all values in result must be values in the domain of the expression.**
  - Basically, tie all tuple variables to a relation

Unsafe queries in TRC

- The following TRC expressions are safe
  - { t(A) | ∃u (R(u) AND u(A) = t(A)) }
  - { t(A) | NOT ∃u (R(u) AND u(A) ≠ t(A)) }
  - { t(A) | ∀u (R(u) => u(A) = t(A)) }

- The following TRC expressions are unsafe
  - { t(A,B) | NOT R(t) }
  - { t(A) | ∃u(u(A) = t(A)) }
  - { t(A) | ∀u(R(u) AND t(A) = 8) }

Domain Relational Calculus (DRC)

- **Uses variables that take values from domains instead of tuples of relations.**

- **If F(d_1, d_2, . . . , d_n) stands for a formula composed of atoms and d_1, d_2, . . . , d_n represent domain variables, then:**

  $\{d_1, d_2, \ldots, d_n \mid F(d_1, d_2, \ldots, d_n)\}$

  is a general domain relational calculus expression.

Example - Domain Relational Calculus

a) Find the names of all managers who earn more than £25,000.

{fN, lN | (∃sN, posn, sex, DOB, sal, bN)
(Staff (sN, fN, lN, posn, sex, DOB, sal, bN) ^
posn = 'Manager' ^ sal > 25000)}

Example - Domain Relational Calculus

b) List the staff who manage properties for rent in Glasgow.

$$\{sN, fN, lN, posn, sex, DOB, sal, bN \mid$$
$$(\exists sN1, cty)(Staff(sN,fN,lN,posn,sex,DOB,sal,bN) \wedge$$
$$(PropertyForRent(pN, st, cty, pc, typ, rms,$$
$$rnt,oN, sN1, bN1) \wedge$$
$$(sN=sN1) \wedge$$
$$cty=\text{‘Glasgow’})\}$$

Example - Domain Relational Calculus

c) List the names of staff who currently do not manage any properties for rent.

$$\{fN, lN \mid (\exists sN)$$
$$(Staff(sN,fN,lN,posn,sex,DOB,sal,bN) \wedge$$
$$(\sim(\exists sN1)\ (PropertyForRent(pN, st, cty, pc, typ,$$
$$rms, rnt,oN, sN1, bN1) \wedge\ (sN = sN1))))\}$$

Note: for brevity, some attributes here were not bound but should have been. See text book p. 106 (Third Edition), p. 108 (Fourth Edition).

You should always bind non-free variables in assignments and exams.

Example - Domain Relational Calculus

d) List the names of clients who have viewed a property for rent in Glasgow.

$$\{fN, lN \mid (\exists cN, cN1, pN, pN1, cty)$$
$$(Client(cN, fN, lN, tel, pT, mR) \wedge$$
$$Viewing(cN1, pN1, dt, cmt) \wedge$$
$$PropertyForRent(pN, st, cty, pc, typ,$$
$$rms, rnt, oN, sN, bN) \wedge$$
$$(cN = cN1) \wedge (pN = pN1) \wedge cty = \text{'Glasgow'})\}$$

Domain Relational Calculus

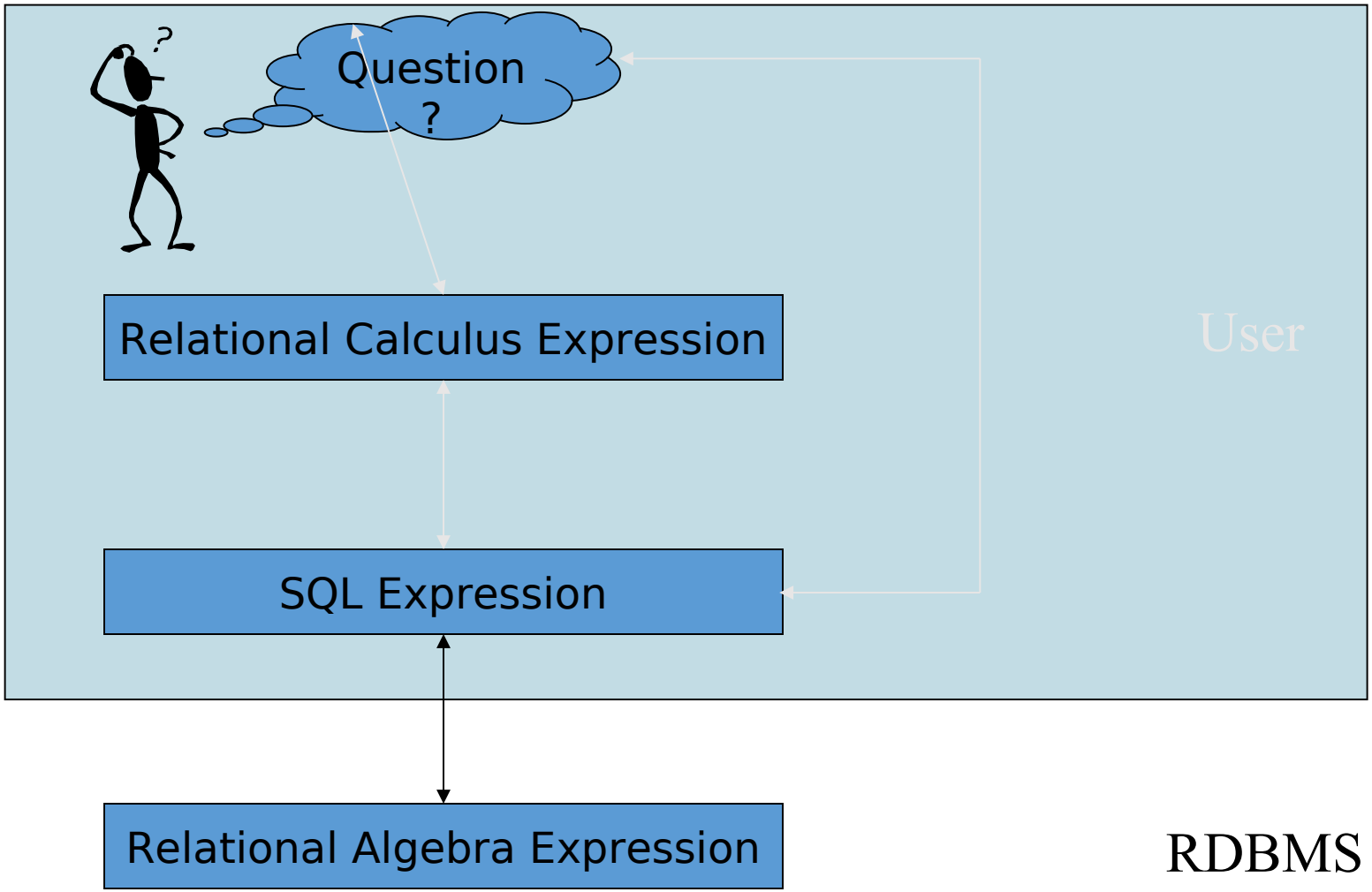
- **When restricted to safe expressions, domain relational calculus is equivalent to tuple relational calculus restricted to safe expressions, which is equivalent to relational algebra.**

- **Means every relational algebra expression has an equivalent relational calculus expression, and vice versa.**

Why Relational Calculus?

- Easy queries can be written in SQL immediately
- Difficult queries require, either:
  - Very much experience; or
  - Trial-and-error iterative approach; or
  - Good understanding of Relational Calculus
- SQL, like RC, is a declarative language
  - With some procedural ingredients (e.g. union)
- Quantifiers in RC are directly translated in SQL (EXISTS)
- Following formal translation algorithms exist:
  - From Calculus to SQL
  - From SQL to Algebra
  - From Algebra to Calculus

Overview of Query Processing



Remarks about the Relational Calculus

- Corresponds to Predicate Logic
  - A.k.a First Order Logic
- Formally, a query is a function mapping a set of relations to a single relation
- Same expressive power as Relational Algebra
- Same theoretical results
- If a query language can express the same queries as the Relational Calculus, then it is relationally complete
- RC, like RA, does not have aggregate functions such as Count, and also not grouping
  - These are extra features provided by SQL
  - Instead, joining can be used for some types of counting

Other Languages

- **Transform-oriented languages are non-procedural languages that use relations to transform input data into required outputs (e.g. SQL).**

- **Graphical languages provide user with picture of the structure of the relation. User fills in example of what is wanted and system returns required data in that format (e.g. QBE).**

Other Languages

- **4GLs can create complete customized application using limited set of commands in a user-friendly, often menu-driven environment.**

- **Some systems accept a form of *natural language*, sometimes called a 5GL, although this development is still a an early stage.**