Chapter 5

Relational Algebra and Relational Calculus

Chapter 5 - 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

- Beside the Structural components, another important part of relational data model is the manipulation mechanism or query language.
- Relational algebra and relational calculus are formal languages associated with the relational model.
- They are used as the basis for other, higher-level Data Manipulation Languages (DMLs) for relational databases.

Introduction

- Informally, relational algebra is a (high-level) procedural language and relational calculus a non-procedural language.
- However, formally both are equivalent to one another.
  - For every expression in the algebra, there is an equivalent expression in the calculus (and vice versa).
- A language that produces a relation that can be derived using relational calculus is relationally complete.

Relational Algebra

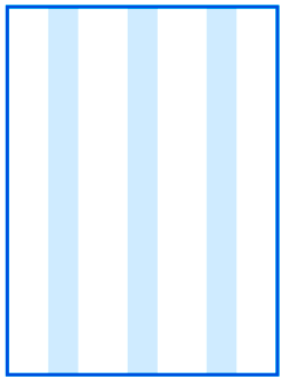
- Relational algebra operations work on one or more relations to define another relation without changing the original relations.
- Both operands and results are relations, so output from one operation can become input to another operation.
- Allows expressions to be nested, just as in arithmetic. This property is called closure.
  - Relations are closed under algebra operations.

Relational Algebra

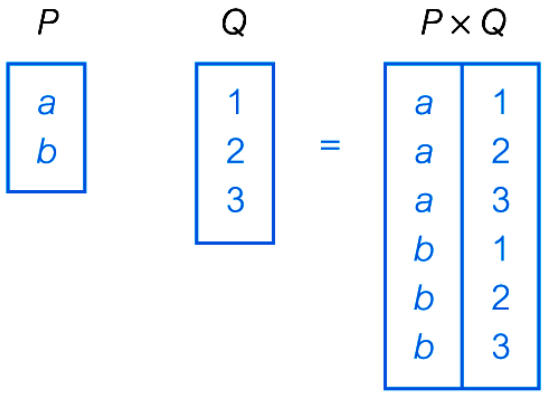
- Five basic operations in relational algebra: Selection, Projection, Cartesian product, Union, and Set Difference.
- These perform most of the data retrieval operations needed.
- Also have Join, Intersection, and Division operations, which can be expressed in terms of 5 basic operations.
- Selection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

Relational Algebra Operations

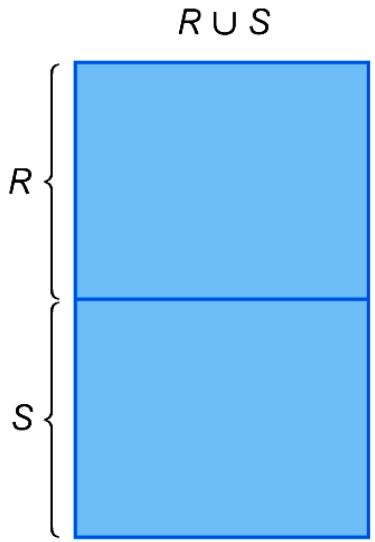
(a) Selection



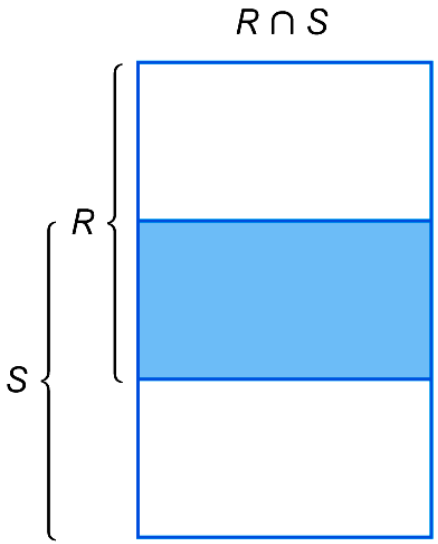
(b) Projection



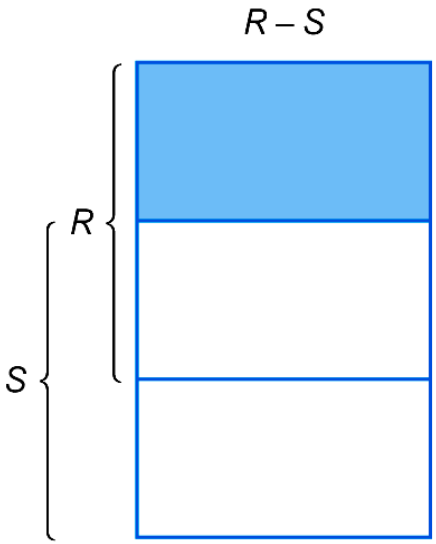
(c) Cartesian product



(d) Union

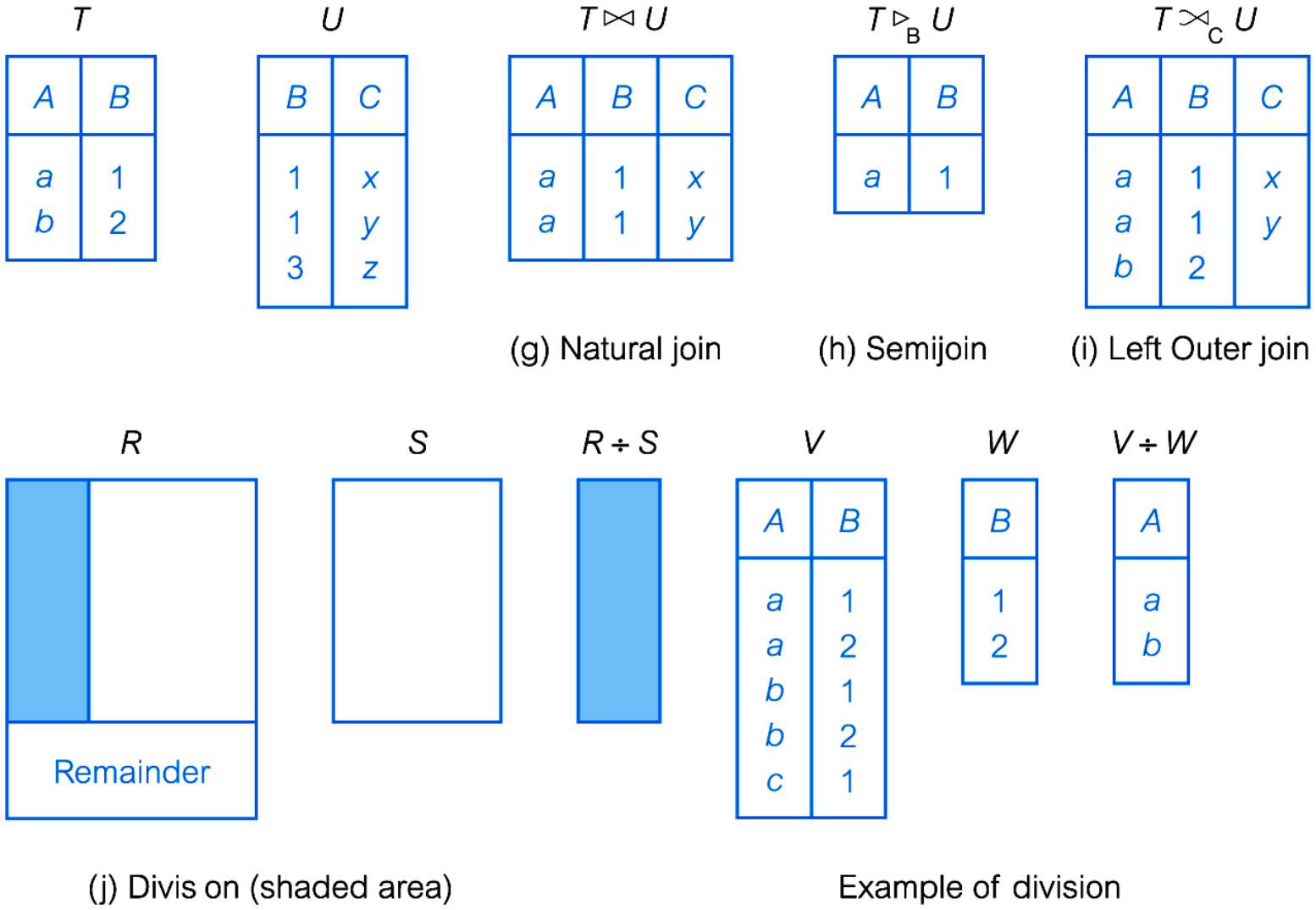


(e) Intersection



(f) Set difference

Relational Algebra Operations



Selection (or Restriction)

- $\sigma_{predicate}$ (R)
  - Works on a single relation R and defines a relation that contains only those tuples (rows) of R that satisfy the specified condition (*predicate*).

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 and defines a relation that contains a vertical subset of R, extracting the values of specified attributes and eliminating duplicates.

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

Union

- R ∪ S
  - 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 – having the same number of attributes with each pair of corresponding attributes having the same domain.
- 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.

Example - Union

- List all cities where there is either a branch office or a property for rent.
- To produce union-compatible relations, we first use the Projection operation

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

city
London
Aberdeen
Glasgow
Bristol

Set Difference

- R – S
  - Defines a relation consisting of the tuples that are in relation R, but not in S.
  - R and S must be union-compatible.

Example - Set Difference

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

$$\Pi_{city}(Branch) - \Pi_{city}(PropertyForRent)$$

city
Bristol

Intersection

- $R \cap S$
  - 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:

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

Example - Intersection

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

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

city
Aberdeen
London
Glasgow

Cartesian product

- R X S
  - Defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.
  - If R has I tuples and N attributes and S has J tuples and M attributes, R X S will contain (I*J) tuples with (N+M) attributes.

Example - Cartesian product

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

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

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.**

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

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

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

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.

$$R \bowtie_F S = \sigma_F(R \times S)$$

- 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.

$$(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \bowtie_{\text{Client.clientNo} = \text{Viewing.clientNo}} (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$$

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

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.

Example - Natural join

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

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

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.**
  - **Natural join displays only rows with matching columns**
- **R ⟕ S**
  - **(Left) outer join is natural join in which tuples from R that do not have matching values in common columns of S are also included in result relation – keeping every tuple in the left relation in the result.**

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

- $R \triangleright_F S$
  - 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:

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

A: set of all attributes for R

Example - Semijoin

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

$$Staff \triangleright_{Staff.branchNo=Branch.branchNo}(\sigma_{city='Glasgow'}(Branch))$$

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 ÷ S**
  - Defines a relation over the attributes C that consists of set of tuples from R that match combination of *every* tuple in S.
  - A divisor table is used to partition a dividend table and produce a quotient or results table. The quotient table is made up of those values of one column for which a second column had all of the values in the divisor.
- **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

Aggregate Operations

- $\mathfrak{I}_{AL}(R)$
  - Applies aggregate function list, AL, to R to define a relation over the aggregate list.
  - AL contains one or more (<aggregate_function>, <attribute>) pairs .
- Main aggregate functions are: COUNT, SUM, AVG, MIN, and MAX.

Example – Aggregate Operations

- How many properties cost more than £350 per month to rent?

$$\rho_R(\text{myCount})\ \Im_{\text{COUNT propertyNo}}\ (\sigma_{\text{rent} > 350}(\text{PropertyForRent}))$$

myCount
5

(a)

Example – Aggregate Operations

- **Find the minimum, maximum, and average staff salary**

$$\rho_R(\text{myMin, myMax, myAverage})$$

$$\Im_{\text{MIN salary, MAX salary, Average salary}} (\text{Staff})$$

myMin	myMax	myAverage
9000	30000	17000

(b)

Grouping Operation

- $_{GA}\Im_{AL}(R)$
  - Groups tuples of R by grouping attributes, GA, and then applies aggregate function list, AL, to define a new relation.
  - AL contains one or more (<aggregate_function>, <attribute>) pairs.
  - Resulting relation contains the grouping attributes, GA, along with results of each of the aggregate functions.

Example – Grouping Operation

- **Find the number of staff working in each branch and the sum of their salaries.**

$$\rho_R(\text{branchNo, myCount, mySum})$$

$$_{\text{branchNo}}\Im_{\text{COUNT staffNo, SUM salary}}(\text{Staff})$$

branchNo	myCount	mySum
B003	3	54000
B005	2	39000
B007	1	9000

Relational Calculus

- Relational calculus query specifies *what* is to be retrieved rather than *how* to retrieve it.
  - No description of how to evaluate a query.
- It takes its name from a branch of symbolic logic called predicate calculus.
  - E.g., to represent "Socrates is mortal" we write mortal (socrates), where mortal is a predicate symbol, and socrates refers to an object.
- When applied to databases, relational calculus has two forms: tuple and domain.

Relational Calculus

- In first-order logic (or predicate), *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' or Staff(x)), 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.
- We may connect predicates by the logical connectives ∧ (AND), ∨ (OR), and (NOT) to form compound predicates.

Tuple Relational Calculus

- 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.
- To specify range (domain) of a tuple variable *S* as the Staff relation, we write:

  Staff(S)

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

  {S | F(S)}

F is called a formula (well-formed formula, or wff)

Tuple Relational Calculus - Example

- To find details (all attributes) of all staff earning more than £10,000:

  $\{S \mid Staff(S) \wedge S.salary > 10000\}$

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

  $\{S.salary \mid Staff(S) \wedge S.salary > 10000\}$

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

$$\text{Staff}(S) \wedge (\exists B)(\text{Branch}(B) \wedge (B.\text{branchNo} = S.\text{branchNo}) \wedge B.\text{city} = \text{‘London’})$$

- **Means ‘There exists a Branch tuple with 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:

  $(\forall B)\ (B.city \neq \text{'Paris'})$

- Means 'For all Branch tuples, the address is not in Paris'.

Tuple Relational Calculus

- Can apply a generalization of De Morgan's laws to existential and universal quantifiers.

$$(\exists X)(F(X)) \equiv \sim (\forall X)(\sim(F(X)))$$

$$(\forall X)(F(X)) \equiv \sim(\exists X)(\sim(F(X)))$$

$$(\exists X)(F_1(X) \wedge F_2(X)) \equiv \sim(\forall X)(\sim(F_1(X)) \vee \sim(F_2(X)))$$

$$(\forall X)(F_1(X) \wedge F_2(X)) \equiv \sim(\exists X)(\sim(F_1(X)) \vee \sim(F_2(X)))$$

- (∀B) (B.city ≠ 'Paris') can also be written as ~(∃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.
- An expression in the tuple relational calculus has the following general form:

$$\{S_1.a_1, S_2.a_2, \ldots, S_n.a_n \mid F(S_1, S_2, \ldots, S_m)\} \quad m \geq n$$

where S_1, S_2, … S_n … S_m are tuple variables; each a_i is an attribute of the relation over which S_i ranges; and F is a formula.

Tuple Relational Calculus

- A (well-formed) formula is made out of one or more atoms having one of the following forms:
  - $R(S_i)$, -- S_i is a tuple variable and R is a relation
  - $S_i.a_1 \; \theta \; S_j.a_2$ -- a_1, a_2: attributes of R
  - $S_i.a_1 \; \theta \; c$ -- θ: comparison operator,
    c: a value of a_1
- 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

- **List the names of all managers who earn more than £25,000.**

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

- **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

- **List the names of staff who currently do not manage any properties.**

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

Or, using DeMorgan's laws:

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

Example - Tuple Relational Calculus

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

$$\{C.fName, C.lName \mid Client(C) \wedge ((\exists V)(\exists P)(Viewing(V) \wedge PropertyForRent(P) \wedge (C.clientNo = V.clientNo) \wedge (V.propertyNo = P.propertyNo) \wedge P.city = \text{'Glasgow'}))\}$$

Example - Tuple Relational Calculus

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

  {B.city | Branch(B) ∧ (~(∃P) (PropertyForRent(P) ∧ (B.city = P.city)))}

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

  {B.city | Branch(B) ∧ ((∃P) (PropertyForRent(P) ∧ (B.city = P.city)))}

Safety of Expressions

- Expressions can generate an infinite set. For example:

  {S | ~Staff(S)}

- To avoid this, add restriction that *all values that appear in the result must be values from the domain of the expression*.
- An expression is safe if the above is true.
- In this example, the domain of the expression is the set of all values appearing in the Staff relation.

Domain Relational Calculus

- Uses variables that take values from domains of attributes 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

- **Find the names of all managers who earn more than £25,000.**

$$\{fN, lN \mid (\exists sN, posn, sex, DOB, sal, bN)$$
$$(Staff(sN, fN, lN, posn, sex, DOB, sal, bN) \wedge$$
$$posn = \text{‘Manager’} \wedge sal > 25000)\}$$

Example - Domain Relational Calculus

- **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

- **List the names of staff who currently do not manage any properties for rent.**

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

Example - Domain Relational Calculus

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

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

Example - Domain Relational Calculus

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

  {cty | (Branch(bN, st, cty, pc) ∨ PropertyForRent(pN, st1, cty, pc1, typ, rms, rnt, oN, sN, bN1))}

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

  {cty | (Branch(bN, st, cty, pc) ∧ (~(∃cty1) PropertyForRent(pN, st1, cty, pc1, typ, rms, rnt, oN, sN, bN1) ∧ (cty=cty1)))}

Example - Domain Relational Calculus

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

  {cty | (Branch(bN, st, cty, pc) ∧
  (∃cty1) PropertyForRent(pN, st1, cty, pc1, typ,
  rms, rnt, oN, sN, bN1) ∧ (cty=cty1)))}

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.
- This means every relational algebra expression has an equivalent relational calculus expression, and vice versa.

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 the user with a picture of the structure of the relation. User fills in an example of what is wanted, and the system returns the 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 (e.g. SQL, QBE).**
- **Some systems accept a form of *natural language*, a restricted version of natural English, sometimes called a 5GL, although this development is still at an early stage.**