Database Management Systems (CSE 220)

Vikas Bajpai

Acknowledgement:

- Pearson Education

Relational Algebra and Relational Calculus

Topics we will cover:

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

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.

Relational Algebra

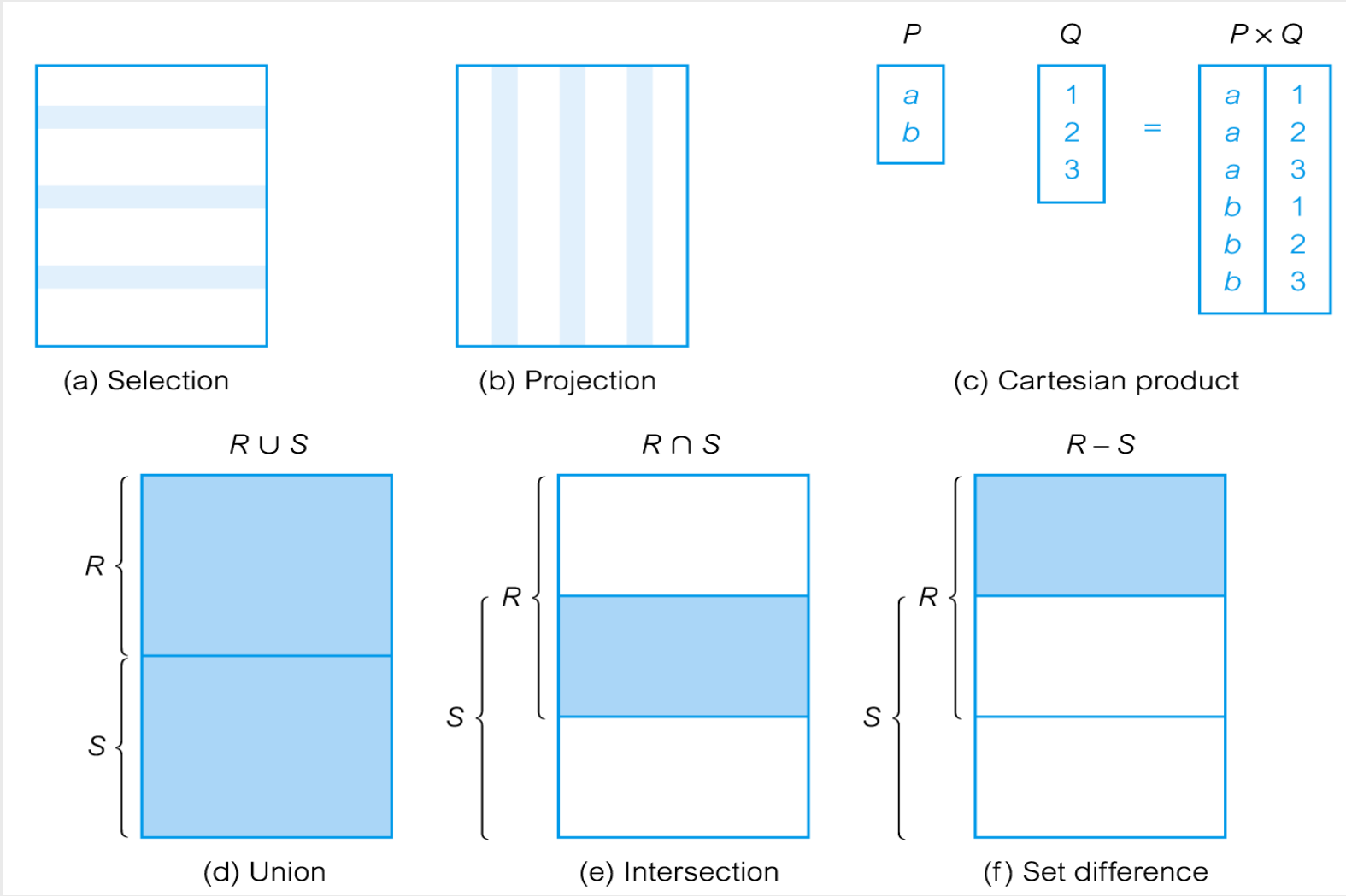
- Relational algebra operations work on one or more relations to define another relation without changing the original relations.
- Both operands and results are relations, so output from one operation can become input to another operation.
- Allows expressions to be nested, just as in arithmetic. This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

Relational Algebra

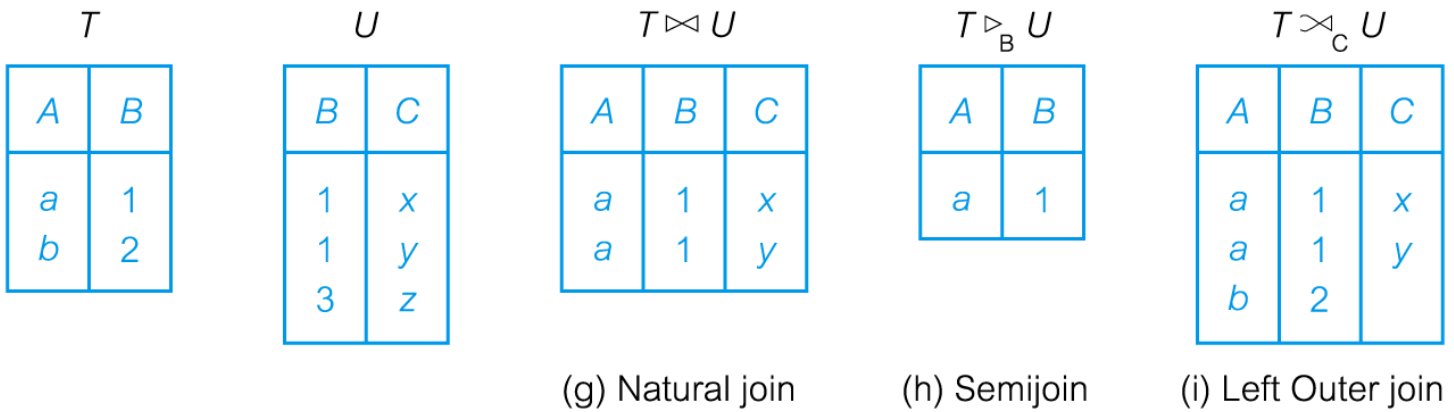
- Five basic operations in relational algebra:
  1. Selection: select rows from a relation
  2. Projection: select column from a relation
  3. Cartesian product
  4. Union
  5. Set Difference.

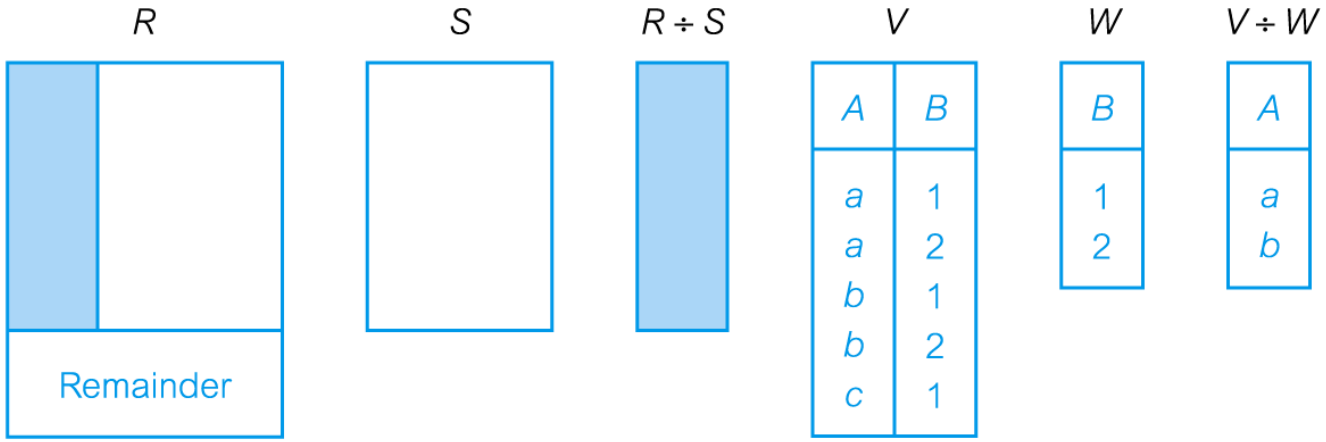
- Also have Join, Intersection, and Division operations, which can be expressed in terms of 5 basic operations.

Relational Algebra Operations



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.

- 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.**

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

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.**

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

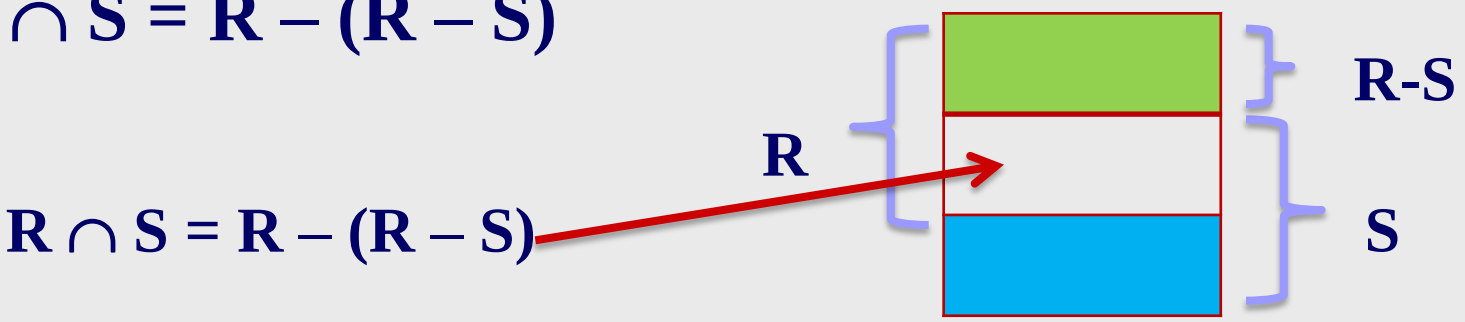
city
Bristol

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.**

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

Id	City
1	Jaipur
2	Jodhpur
3	Jaisalmer
4	Kota

$\cap$

Id	City
1	Delhi
2	Jaipur
3	Chennai
4	Kota

$=$

City
Jaipur
Kota

Cartesian product (R X S)

- Defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.
- The Cartesian Product is also an operator which works on two sets.
- It is sometimes called the CROSS PRODUCT or CROSS JOIN.
- It combines the tuples of one relation with all the tuples of the other relation.

Example - Cartesian product (R X S)

R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4

R CROSS S

A	1	A	1
A	1	C	2
A	1	D	3
A	1	E	4
B	2	A	1
B	2	C	2
B	2	D	3
B	2	E	4
D	3	A	1
D	3	C	2
D	3	D	3
D	3	E	4

F	4	A	1
F	4	C	2
F	4	D	3
F	4	E	4
E	5	A	1
E	5	C	2
E	5	D	3
E	5	E	4

Example - Cartesian product (R X S)

Id	City
1	Jaipur
2	Jodhpur
3	Jaisalmer

x

Id	City
1	Delhi
2	Jaipur
3	Chennai
4	Kota

=

Id	City	Id	City
1	Jaipur	1	Delhi
1	Jaipur	2	Jaipur
1	Jaipur	3	Chennai
1	Jaipur	4	Kota
2	Jodhpur	1	Delhi
2	Jodhpur	2	Jaipur
2	Jodhpur	3	Chennai
2	Jodhpur	4	Kota
3	Jaisalmer	1	Delhi
3	Jaisalmer	2	Jaipur
3	Jaisalmer	3	Chennai
3	Jaisalmer	4	Kota

Example - Cartesian product and Selection

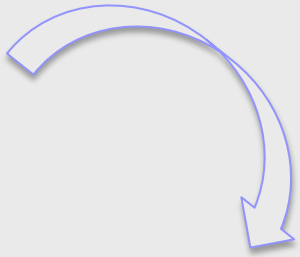
- **Use selection operation to extract those tuples where Branch.Id= PropertyForRent.Id.**

$$\Pi_{\text{Branch.Id, PropertyForRent.City}} (\sigma_{\text{Branch.Id= PropertyForRent.Id}} (\text{Branch X PropertyForRent}))$$

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

Example - Cartesian product and Selection

Id	City	Id	City
1	Jaipur	1	Delhi
1	Jaipur	2	Jaipur
1	Jaipur	3	Chennai
1	Jaipur	4	Kota
2	Jodhpur	1	Delhi
2	Jodhpur	2	Jaipur
2	Jodhpur	3	Chennai
2	Jodhpur	4	Kota
3	Jaisalmer	1	Delhi
3	Jaisalmer	2	Jaipur
3	Jaisalmer	3	Chennai
3	Jaisalmer	4	Kota



Id	City
1	Delhi
2	Jaipur
3	Chennai

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)

- $\mathbf{R} \bowtie_{\mathbf{F}} \mathbf{S}$
  - 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 ($<, \leq, >, \geq, =, \neq$).
- Produces all possible combinations of tuples from R_1 and R_2 that satisfy the join condition.

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.

Equijoin (a particular type of Theta join)

- Produces all the combinations of tuples from relations R_1 and R_2 that satisfy a join condition with only equality comparisons.

In other words:

- When Theta join uses only **equality** comparison operator, it is said to be equijoin.

Sample Tables

Prof

Fac_ID	Name	Dept	Tax	Rank
091	Abhinav	CSE	20000	Lecturer
101	Diwakar	CSE	30000	Asso Prof
111	Shashank	ECE	25000	Asst Prof
123	Vivek	ECE	35000	Prof

Course

Code	Title	Fac_ID
C220	DBMS	091
C222	OS	101
C123	POC	123
C243	BE	111

Example - Equijoin

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

$$\Pi_{\text{Name, Dept}}(\text{Prof}) \bowtie_{\text{Prof.Faculty\_ID = Course.Faculty\_ID}} (\Pi_{\text{Code, Title}}(\text{Courses}))$$

Name	Dept	Code	Title
Abhinav	CSE	C220	DBMS
Diwakar	CSE	C222	OS
Shashank	ECE	C123	POC
Vivek	ECE	C243	BE

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.
  - We can perform a Natural Join only if there is at least one common attribute that exists between two relations

Sample Tables

Prof

Fac_ID	Name	Dept	Tax	Rank
091	Abhinav	CSE	20000	Lecturer
101	Diwakar	CSE	30000	Asso Prof
111	Shashank	ECE	25000	Asst Prof
123	Vivek	ECE	35000	Prof

Course

Code	Title	Fac_ID
C220	DBMS	091
C222	OS	101
C123	POC	123
C243	BE	111

Example - Natural join

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

$$(\Pi_{\text{Fac\_ID, Name, Dept, Tax, Rank, Title}}(\text{Prof})) \bowtie (\Pi_{\text{Code, Title, Fac\_ID}}(\text{Course}))$$

Fac_ID	Name	Dept	Tax	Rank	Code	Title
091	Abhinav	CSE	20000	Lecturer	C220	DBMS
101	Diwakar	CSE	30000	Asso Prof	C222	OS
111	Shashank	ECE	25000	Asst Prof	C123	POC
123	Vivek	ECE	35000	Prof	C243	BE

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.

Example - Left Outer join

- Produce a status report on property viewings.

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

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:**

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

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.**

- **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

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

Guest_No	Room_No
20	100
30	200
30	300
10	100
30	100
20	300

÷

Room_No
100
200
300

=

Guest_No
30

Division (R ÷ S): Formal Derivation

- **Expressed using basic operations:**

$T_1 \leftarrow \Pi_C(R)$ // Restrict T1 to the attributes in R that are not in S

$T_2 \leftarrow \Pi_C((T_1 X S) - R)$ // Remove from R the multi-rows that match S

$T \leftarrow T_1 - T_2$ // Remove from R the rows that do not fully match the multi-rows in S

R

X	Y
a	1
a	2
b	1
b	2
c	1

S

Y
1
2

T_1

X
a
b
c

T_1 X S

X	Y
a	1
a	2
b	1
b	2
c	1
c	2

T_2

X
c

T

X
a
b

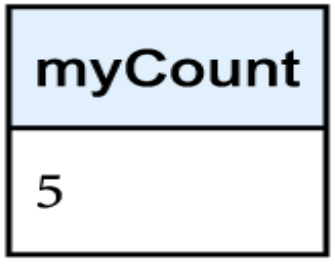
Aggregate Operations

- $\mathfrak{I}_{AL}(R)$
  - 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?**

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



(a)

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.**
- **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 forms: *tuple* and *domain*.**

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.**
- **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:**

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

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

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

Tuple Relational Calculus – Relation with SQL

- If you have trouble writing a relational calculus query, write an SQL query and translate it:

  To find the salary details of all staff earning more than £10,000:

  –SQL: SELECT S.staffNo, S.salary

  FROM Staff S WHERE S.salary > 10000

  –Relational Calculus:

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

  –Translation

  »Select attributes go on left side of | character

  »FROM relations are mapped to tuple variables and combined using AND

  »WHERE is appended to tuple variables with an AND

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:**

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

- **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 $\neq$ 'Paris')

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

- **Can also use $\sim(\exists B)$ (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 atoms:**
  - » $R(S_i)$, 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.

Tuple Relational Calculus

- Boolean operators are typically used for SELECT queries
- (∃ X)(F) is typically used for joins
- (∃ X)(F) and (∀ X)(F) are typically used for integrity constraints
  - Example: All staff members must make less than $10000
  - Example: There does not exist a staff member who manages more than 100 properties.

Example - Tuple Relational Calculus

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

{S.fName, S.lName | Staff(S) ∧

S.position = ‘Manager’ ∧ S.salary > 25000}

Example: Relational Calculus Queries

- **List the staff who manage properties for rent in Glasgow.**

{S | Staff(S) ∧ (∃P) (PropertyForRent(P) ∧ (P.staffNo = S.staffNo) ∧ P.city = 'Glasgow')}

SQL: SELECT * FROM Staff S

WHERE EXISTS (SELECT * FROM PropertForRent P

WHERE P.staffNo = S.staffNo

AND P.city = '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

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

Example - Tuple Relational Calculus

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

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

Tuple Relational Calculus

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

- **To avoid this, add restriction that all values in result must be values in the domain of the expression.**

Domain Relational Calculus

- **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

- **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.**

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

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.

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 at an early stage.**