Chapter 5

Relational Algebra and Relational Calculus

Overview

- The previous chapter covers the relational model, which provides a formal description of the structure of a database
- This chapter covers the relational algebra and calculus, which provides a formal basis for the query language for the database

Chapter 5 – 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 declarative, 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.**

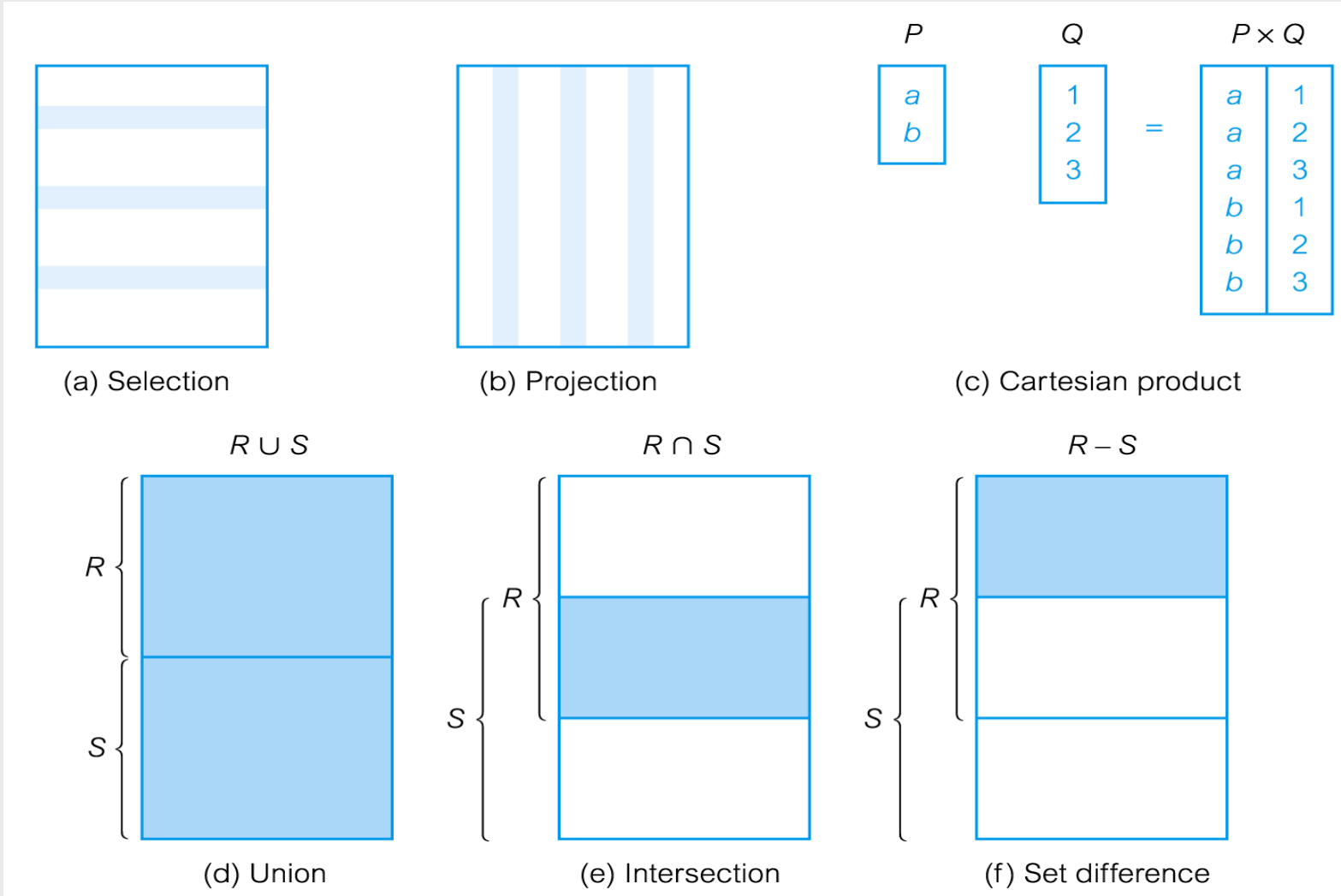
Relational Algebra

- **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.**

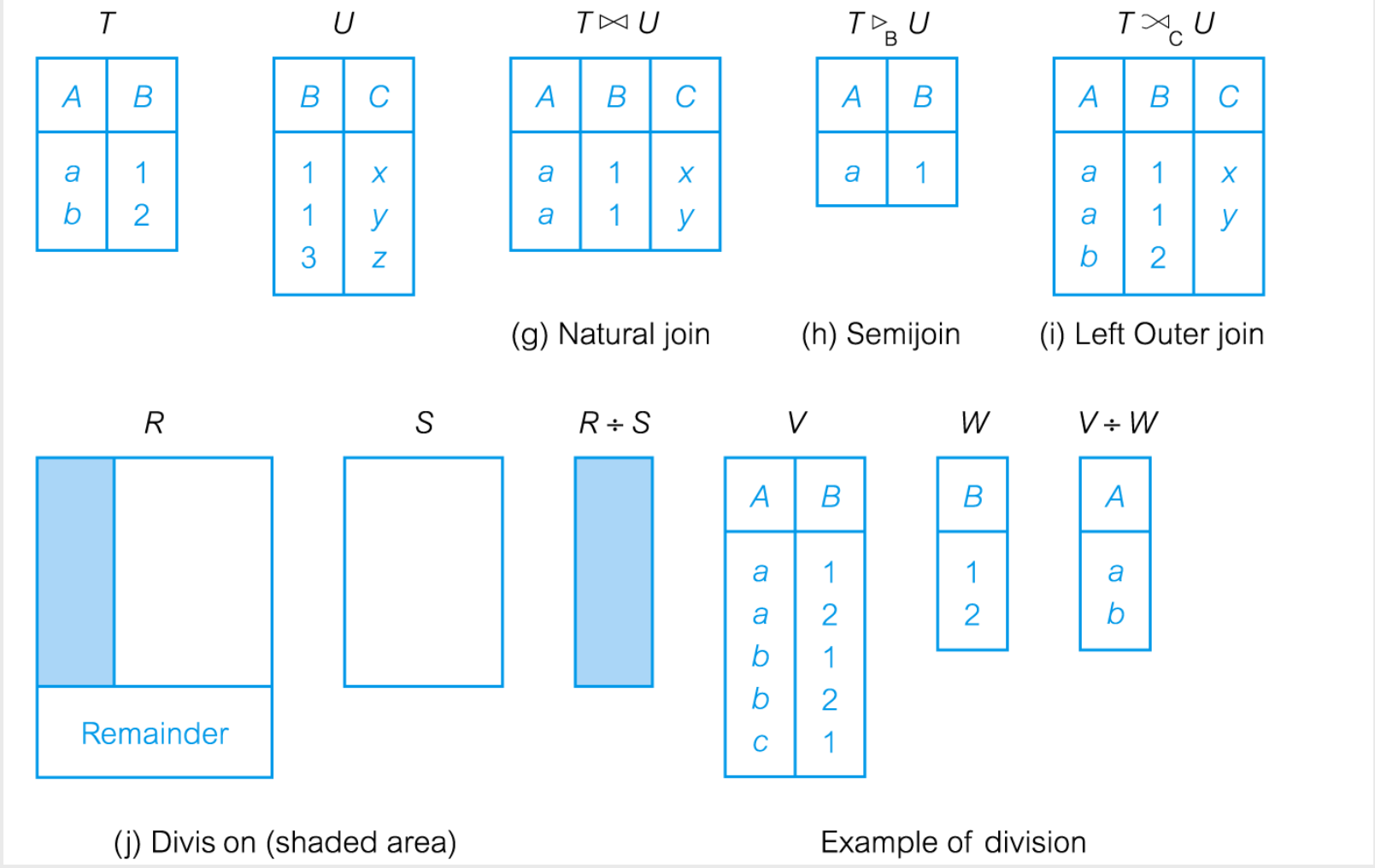
Relational Algebra

- Five basic operations in relational algebra:
  1. Selection: selects rows from a relation
  2. Projection: selects columns 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 ($R \cup S$)

- defines a relation that contains all the tuples of R, or S, or both R and S
  - duplicate tuples are eliminated.
  - R and S must be "union compatible", which means they must have the same set of named attributes (the attributes may appear in two different orders as long as the names can be matched).
- 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.**

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

Id	City	Address
1	Knoxville	5485 Alcoa Hwy
2	Columbus	1678 Cardiff Rd.
3	Pittsburgh	100 Rockwood Ave
4	Columbus	585 Tremont Rd.

$\cup$

PropId	City	Address
1	Nashville	110 Parthenon Way
2	Nashville	4868 Vanderbilt Dr
3	Knoxville	2408 Trillium Ln
4	Atlanta	5868 Peachtree Rd
5	Pittsburgh	386 Cedar Ln.

=

City
Knoxville
Columbus
Pittsburgh
Nashville
Atlanta

Note that we made the two relations "union-compatible" by projecting them down to a common column

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.**

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

Id	City	Address
1	Knoxville	5485 Alcoa Hwy
2	Columbus	1678 Cardiff Rd.
3	Pittsburgh	100 Rockwood Ave
4	Columbus	585 Tremont Rd.

−

PropId	City	Address
1	Nashville	110 Parthenon Way
2	Nashville	4868 Vanderbilt Dr
3	Knoxville	2408 Trillium Ln
4	Atlanta	5868 Peachtree Rd
5	Pittsburgh	386 Cedar Ln.

=

City
Columbus

Note that we made the two relations "union-compatible" by projecting them down to a common column

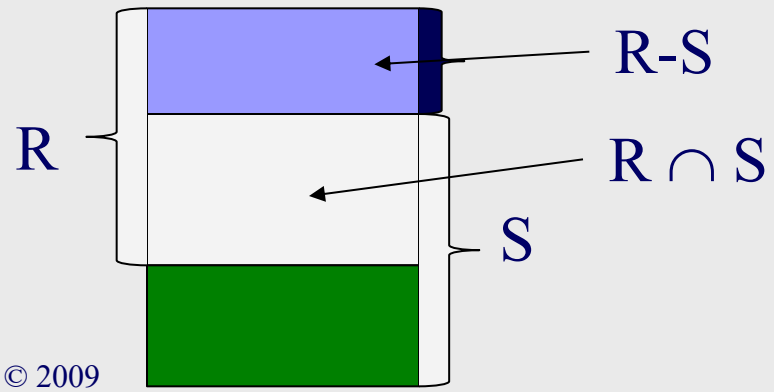
Intersection

- **R ∩ 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:**

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

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



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	Address
1	Knoxville	5485 Alcoa Hwy
2	Columbus	1678 Cardiff Rd.
3	Pittsburgh	100 Rockwood Ave
4	Columbus	585 Tremont Rd.

$\cap$

PropId	City	Address
1	Nashville	110 Parthenon Way
2	Nashville	4868 Vanderbilt Dr
3	Knoxville	2408 Trillium Ln
4	Atlanta	5868 Peachtree Rd
5	Pittsburgh	386 Cedar Ln.

$=$

City
Knoxville
Pittsburgh

Note that we made the two relations "union-compatible" by projecting them down to a common column

Cartesian product

- **R X S**
  - **Defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.**

Example - Cartesian product

(Student) X (Courses)

Id	Gpa
1	3.5
2	2.6

X

Id	CourseId
1	CS140
1	ECE255
2	CS302

=

Id	Gpa	Id	CourseId
1	3.5	1	CS140
1	3.5	1	ECE255
1	3.5	2	CS302
2	2.6	1	CS140
2	2.6	1	ECE255
2	2.6	2	CS302

Example - Cartesian product and Selection

- **Use selection operation to extract those tuples where Student.Id= Courses.Id.**

  $\Pi_{\text{student.Id, CourseId}}(\sigma_{\text{Student.Id= Courses.Id}}(\text{Student X Courses}))$

Id	Gpa	Id	CourseId
1	3.5	1	CS140
1	3.5	1	ECE255
1	3.5	2	CS302
2	2.6	1	CS140
2	2.6	1	ECE255
2	2.6	2	CS302

=>

Id	courseId
1	CS140
1	ECE255
2	CS302

- **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 ($<, \leq, >, \geq, =, \neq$).

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 students who have taken a course.**

$$(\Pi_{Id,\ Name}(Student)) \bowtie_{Student.Id = Courses.Id} (\Pi_{Id,\ CourseId,\ Comment}(Courses))$$

Id	Name
1	Smiley
2	Pooh
3	Nels

$\bowtie$

Id	CourseId	Comment
1	CS140	"Great course"
1	ECE255	"Amazing"
2	CS302	"Crushed my gpa"

=

Id	Name	Id	CourseId	Comment
1	Smiley	1	CS140	"Great course"
1	Smiley	1	ECE255	"Amazing"
2	Pooh	2	CS302	"Crushed my gpa"

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 students who have taken a course.**

$(\Pi_{Id,\ Name}(Student)) \bowtie (\Pi_{Id,\ CourseId,\ Comment}(Courses))$

Id	Name
1	Smiley
2	Pooh
3	Nels

$\bowtie$

Id	CourseId	Comment
1	CS140	"Great course"
1	ECE255	"Amazing"
2	CS302	"Crushed my gpa"

=

Id	Name	CourseId	Comment
1	Smiley	CS140	"Great course"
1	Smiley	ECE255	"Amazing"
2	Pooh	CS302	"Crushed my gpa"

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 report that shows all students and the courses they are taking, even if the student is not taking a course.**

$$(\Pi_{Id,\ Name}(Student)) \ ⟕_{Student.Id=Courses.id} \ (\Pi_{Id,\ CourseId,\ Comment}(Courses))$$

Id	Name
1	Smiley
2	Pooh
3	Nels

⟕

Id	CourseId	Comment
1	CS140	“Great course”
1	ECE255	“Amazing”
2	CS302	“Crushed my gpa”

=

Id	Name	CourseId	Comment
1	Smiley	CS140	“Great course”
1	Smiley	ECE255	“Amazing”
2	Pooh	CS302	“Crushed my gpa”
3	Nels	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 students who are taking CS140.**

$$\text{Student} \triangleright_{\text{Student.Id=Courses.Id}} (\sigma_{\text{courseId= 'CS140'}} (\text{Courses}))$$

Id	Name
1	Smiley
2	Pooh
3	Nels

$\triangleright$

Id	CourseId	Comment
1	CS140	"Great course"
1	ECE255	"Amazing"
2	CS140	"Crushed my gpa"

$=$

Id	Name
1	Smiley
2	Pooh

Division (R ÷ S)

- **Example Queries**
  - **List all clients who have viewed all properties with three rooms**
  - **List all guests who have stayed in all rooms of the Grosvenor Hotel (the relations below were obtained from the booking and room relations by selecting only those bookings and rooms associated with the Grosvenor Hotel and by projecting away the dateFrom, bookingNo and dateTo fields in the booking relation)**

guestNo	roomNo
20	100
30	200
30	300
10	100
30	100
20	300

÷

roomNo
100
200
300

=

guestNo
30

Division (R ÷ S)

- **Preconditions**
  - **R is defined over the attribute set A**
  - **S is defined over the attribute set B, such that $B \subseteq A$**
  - **Let C = A – B (i.e., C is the set of attributes of R that are not attributes of 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 more "severe" form of a semi-join, because instead of a match applying to a single row in R and a single row in S, it requires a match between multiple rows of R and S**

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

$T_2 \leftarrow \Pi_C((T_1 X\ S) - R)$ // T2 contains the "fragment" tuples from R that do not match all rows in 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

- ρ_R(colnames) $\mathfrak{I}_{AL}$(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 .
  - ρ_R(colnames) names the columns being created by the aggregate functions: same as a projection
- 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)

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*.
  - In this class only consider tuple relational calculus

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 using the notation:**

  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 | Staff(S) ∧ 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**

Cartesian Product

The relational calculus expression for R X S is

{R, S | R(T1) ∧ S(T2)}

Join

The relationl calculus expression for

R(a,b) Join S(b, c) is

{R, S | R(T1) ∧ S(T2) ∧ (T1.b = T2.b)}

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.**
  - **Only variables to the left of the bar (|) may be free variables**

Tuple Relational Calculus

- **Example: List all tuples in Staff such that the staff member works in a London Branch:**
  - **Relational calculus query**

    $\{S.fName, S.lName \mid Staff(S) \wedge (\exists B)(Branch(B) \wedge (B.branchNo = S.branchNo) \wedge B.city = \text{'London'})\}$
- **This example is a semi-join of Staff and Branch**

Tuple Relational Calculus

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

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

- Means 'For all Branch tuples, the address is not in Paris'.
- Can also use $\sim(\exists B)\ (Branch(B) \wedge B.city = \text{'Paris'})$ 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
  - » $S_i.a_1 \ \theta \ S_j.a_2$ where θ is a relational or Boolean operator
  - » $S_i.a_1 \ \theta \ c$
- **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
- $(\exists X)(F)$ is typically used for semi-joins
- $\sim(\exists X)(F)$ and $(\forall X)(F)$ are typically used for integrity constraints
  - Example: All staff members must make less than $100,000
  - Example: There does not exist a staff member who manages more than 100 properties
- $\exists$ and $\forall$ form subqueries

Tuple Relational Calculus: Set Operations

Union: List all cities where there is either a branch office or a property for rent:

$$B.city \cup P.city = \{ T.city \mid Branch(T) \vee PropertyForRent(\mathrm{T}) \}$$

Difference: List all cities where there is a branch office but no property for rent:

$$B.city - P.city = \{B \mid Branch(B) \wedge \sim (\exists P)(\mathrm{Pr}\, opertyFor\, \mathrm{Re}\, nt(P) \wedge B.city = P.city)\}$$

Intersection: List all cities where there is both a branch office and a property for rent:

$$B.city \cap P.city = \{B.city \mid Branch(B) \wedge (\exists P)(\mathrm{Pr}\, opertyFor\, \mathrm{Re}\, nt(P) \wedge B.city = P.city)\}$$

Tuple Relational Calculus: Set Division

- Suppose you have the following relations:
  - Tourist(tname, addr): tourist information
  - Visit(tname, pname): parks visited by tourists
  - Park(pname, state): park information
- List the tourists who have visited all parks in TN

$$\{T \mid Tourist(T) \wedge (\forall P)(Park(P)$$
$$\wedge((P.state != 'TN') \vee (\exists V)(Visit(V) \wedge T.tname = V.tname$$
$$\wedge V.pname = P.pname)))\}$$

Tuple Relational Calculus: Left Outer Join

Left Outer Join

- Suppose you have the following relations
  - Student(id, name, addr): student information
  - Enrollment(id, coursed): course enrollment information
  - List the students and courses they are taking and include all students in the result, even if they are not taking a course
  - $\{S.name, E.courseID \mid Student(S) \wedge Enrollment(E) \wedge S.id = E.id) \vee (Student(S) \wedge \sim(\exists E)(Enrollment(E) \wedge S.id = E.id)\}$

Example - Tuple Relational Calculus

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

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

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 S.* FROM Staff S

WHERE EXISTS(SELECT * FROM PropertyForRent P

WHERE P.staffNo = S.staffNo

AND P.city = 'Glasgow');

Example Relational Calculus Queries

- **List the staff who manage properties for rent in Glasgow.**
  - **Seemingly comparable Query that you might try which is wrong**

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

  - PropertyForRent(P) makes P a free variable, even though P does not appear on the left side of |

Example Relational Calculus Queries

- **List each property that rents for more than $450 and list the name of the staff member who manages the property.**

$$\{P, S.name \mid Staff(S) \wedge PropertyForRent(P) \wedge (P.staffNo = S.staffNo) \wedge P.rent > 450)\}$$

SQL: SELECT P.*, S.name FROM Staff S Natural Join PropertyForRent P

WHERE P.rent > 450;

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

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

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.**