DAMG 6210
Database Design

Nik Bear Brown
@NikBearBrown

Relational Algebra

Relational Calculus

Topics

- Relational Algebra
- Relational Calculus

Relational algebra and relational calculus

- 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 and relational calculus

Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:

Relational Algebra: More operational, very useful for representing execution plans.

Relational Calculus: Lets users describe what they want, rather than how to compute it. (Non-operational, *declarative*.)

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: Selection, Projection, Cartesian product, Union, and Set Difference.

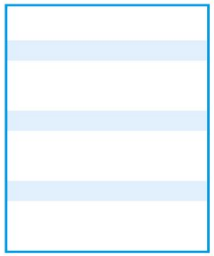
- These perform most of the data retrieval operations needed.

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

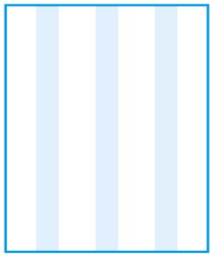
Relational Algebra

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* (-) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 and in reln. 2.
- Additional operations:
  - Intersection, *join*, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be *composed*! (Algebra is "closed".)

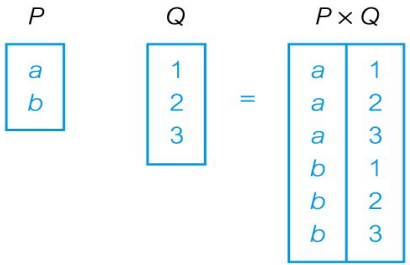
Relational Algebra Operations



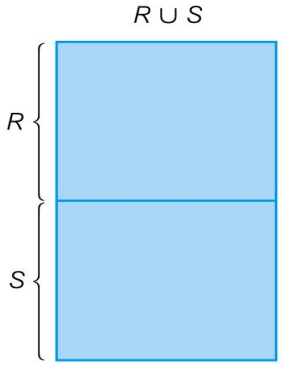
(a) Selection



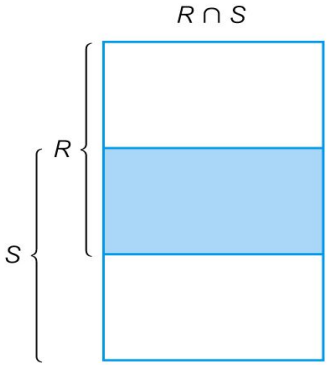
(b) Projection



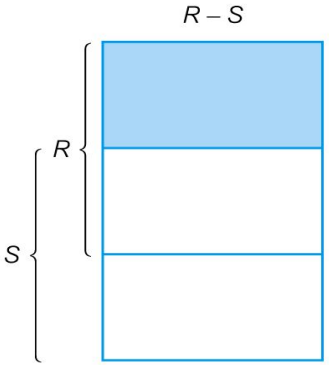
(c) Cartesian product



(d) Union

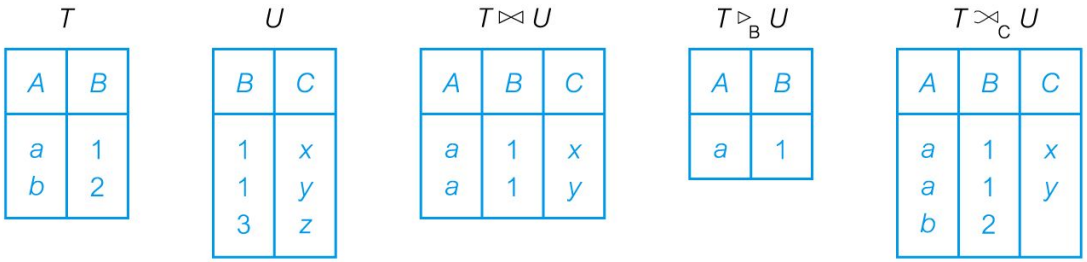


(e) Intersection



(f) Set difference

Relational Algebra Operations



T

A	B
a	1
b	2

U

B	C
1	x
1	y
3	z

$T \bowtie U$

A	B	C
a	1	x
a	1	y

(g) Natural join

$T \triangleright_B U$

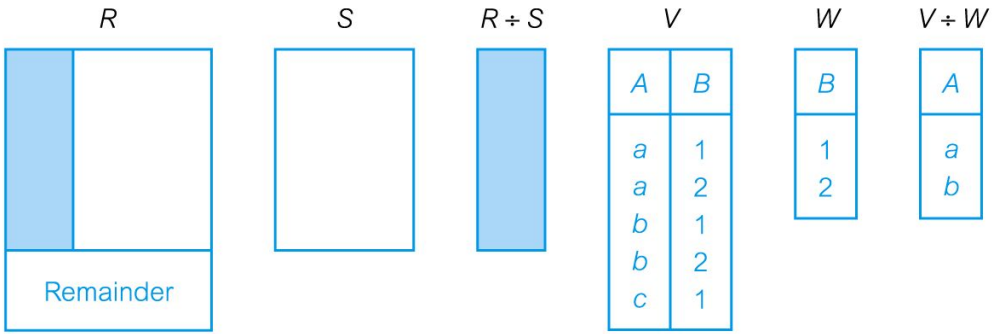
A	B
a	1

(h) Semijoin

$T \rtimes_C U$

A	B	C
a	1	x
a	1	y
b	2	

(i) Left Outer join



(j) Divis on (shaded area)

V

A	B
a	1
a	2
b	1
b	2
c	1

W

B
1
2

V ÷ W

A
a
b

Example of division

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

Selection

- Selects rows that satisfy *selection condition*.
- No duplicates in result! (Why?)
- *Schema* of result identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$$\sigma_{rating>8}(S2)$$

sname	rating
yuppy	9
rusty	10

$$\pi_{sname,rating}(\sigma_{rating>8}(S2))$$

Selection

- Notation: $\sigma_{p}(r)$
- *p* is called the selection predicate , **r** can be the name of a table, or another query
- Predicate:
  1. Simple
     - attr1 = attr2
     - Attr = constant value
       - (also, <, > , etc)
  2. Complex
     - predicate1 AND predicate2
     - predicate1 OR predicate2
     - NOT (predicate)

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.

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

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

Projection

- Deletes attributes that are not in *projection list*.
- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate *duplicates*! (Why??)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$$\pi_{sname,rating}(S2)$$

age
35.0
55.5

$$\pi_{age}(S2)$$

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.

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

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.

Cartesian-Product

- **S1 × R1: Each row of S1 paired with each row of R1.**

  Like the c.p for mathematical relations: every tuple of S1 "appended" to every tuple of R1

- Q: How many rows in the result?
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.
  - *May have a naming conflict*: Both S1 and R1 have a field with the same name.
  - In this case, can use the *renaming operator*…

Cartesian Product Example

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

R1

S1 X R1 =

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/ 10/ 96
22	dustin	7	45.0	58	103	11/ 12/ 96
31	lubber	8	55.5	22	101	10/ 10/ 96
31	lubber	8	55.5	58	103	11/ 12/ 96
58	rusty	10	35.0	22	101	10/ 10/ 96
58	rusty	10	35.0	58	103	11/ 12/ 96

Cartesian Products

- For sets A, B, the Cartesian product, or cross product, of A and B is denoted by $A \times B$ and equals $\{(a, b) \mid a \in A, b \in B\}$
- Elements of $A \times B$ are **ordered pairs**. For (a, b), (c, d) $\in A \times B$, (a, b) = (c, d) if and only if a = c and b = d

Cartesian Products

Properties:

1. If A, B are finite, it follows from the rule of product that $|A \times B| = |A||B|$
2. Although we generally will not have $A \times B = B \times A$, we will have $|A \times B| = |B \times A|$

Cartesian Products Example A

Let $A = \{2, 3, 4\}$, $B = \{4, 5\}$. Then

a) $A \times B = \{(2, 4), (2, 5), (3, 4), (3, 5), (4, 4), (4, 5)\}$

b) $B \times A = \{(4, 2), (4, 3), (4, 4), (5, 2), (5, 3), (5, 4)\}$

c) $B^2 = B \times B = \{(4, 4), (4, 5), (5, 4), (5, 5)\}$

d) $B^3 = B \times B \times B = \{(a, b, c) \mid a, b, c \in B\}$; for instance, $(4, 5, 5) \in B^3$

Cartesian Products Example B

An experiment E is conducted as follows:

A single dice is rolled and its outcome noted,

and then a coin is flipped and its outcome noted.

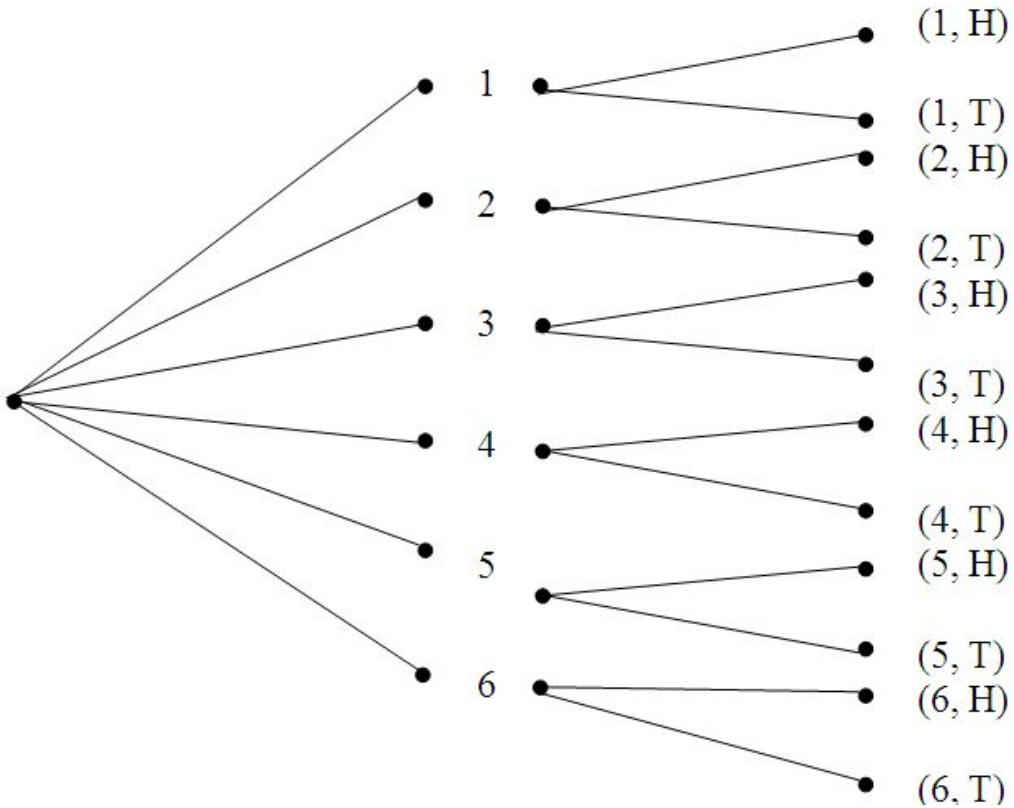
Determine a sample space S for E

S1={1, 2, 3, 4, 5, 6} be a sample space dice.

S2= {*H, T}* be a sample space coin.

Then S = S1 × S2 is a sample space for E.

Cartesian Products Example B



Relations

Let A = {0,1,2}, B = {1,2,3}. A x B = {(0,1), (0,2), (0,3), (1,1), (1,2), (1,3), (2,1), (2,2), (2,3)}

Let say an element x in A is related to an element y in B iff x is less than y. x R y: x is related to y

∴0 R 1, 0 R 2, 0 R 3, 1 R 2, 1 R 3, 2 R 3

∴The set of all ordered pair in A x B where elements are related {(0,1), (0,2), (0,3), (1,2), (1,3), (2,3)}

Relations

- For sets A, B, a (binary) relation R from A to B is a subset of $A \times B$. Any subset of $A \times A$ is called a (binary) relation on A
- Given an ordered pair (a, b) in A x B, x is related to y by R (x R y) iff (x, y) is in R
- In general, for finite sets A, B with $|A| = m$ and $|B| = n$, there are 2^{mn} relations from A to B, including the empty relation as well as the relation $A \times B$ itself

Relations Example

Let A = {2, 3, 4}, B = {4, 5}. Then

A × B = {(2, 4), (2, 5), (3, 4), (3, 5), (4, 4), (4, 5)}.

The following are some of the relations from A to B.

i. ∅

ii. {(2, 4)}

iii. {(2, 4), (2, 5)}

iv. {(2, 4), (3, 4), (4, 4)}

v. {(2, 4), (3, 4), (4, 5)}

vi. A × B

Since |A × B| = 6, there are 2^6 possible relations from A to B (for there are 2^6 possible subsets of A × B)

Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - `Corresponding' fields have the same type.
- What is the *schema* of result?

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

Cross-Product

- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.
  - *Conflict*: Both S1 and R1 have a field called *sid*.

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/ 10/ 96
22	dustin	7	45.0	58	103	11/ 12/ 96
31	lubber	8	55.5	22	101	10/ 10/ 96
31	lubber	8	55.5	58	103	11/ 12/ 96
58	rusty	10	35.0	22	101	10/ 10/ 96
58	rusty	10	35.0	58	103	11/ 12/ 96

$$\rho\,(C(1 \to sid1, 5 \to sid2),\ S1 \times R1)$$

Example - Cartesian product and Selection

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

$$s_{Client.clientNo = Viewing.clientNo}((Õ_{clientNo, fName, lName}(Client)) X (Õ_{clientNo, propertyNo, comment}(Viewing)))$$

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

Joins

- *Condition Join*: $R \bowtie_c S = \sigma_c (R \times S)$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/ 12/ 96
31	lubber	8	55.5	58	103	11/ 12/ 96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*.

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 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.

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

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.

- 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.

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

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

- $R \div S$
  - 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:

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

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

$$T \leftarrow T_1 - T_2$$

Expressing A/B Using Basic Operators

- Division is not essential op; just a useful shorthand.
  - (Also true of joins, but joins are so common that systems implement joins specially.)
- *Idea*: For *A/B*, compute all *x* values that are not `disqualified' by some *y* value in *B*.
  - *x* value is *disqualified* if by attaching *y* value from *B*, we obtain an *xy* tuple that is not in *A*.

Disqualified x values: $\pi_x((\pi_x(A) \times B) - A)$

A/B: $\pi_x(A)$ − Disqualified x values

Example - Division

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

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

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

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

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

propertyNo
PG4
PG36

RESULT

clientNo
CR56

Aggregate Operations

- $\Im_{AL}(R)$
  - 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)

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.

$$\rho_R(\text{branchNo}, \text{myCount}, \text{mySum})$$
$$_{\text{branchNo}}\mathcal{F}_{\text{COUNT staffNo, SUM salary}}(\text{Staff})$$

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