Relational Algebra and Relational Calculus

Relational Algebra and Relational Calculus

- Three components of Relational Data Model:
  - a structural part;
  - a set of integrity rules;
  - a manipulative part (Query Languages).

Formal Relational Query Languages

- Two mathematical Query Languages form the basis for "real" languages (e.g. SQL):
  - *Relational Algebra*:
  - Procedural language
  - More operational, very useful for representing execution plans
  - *Relational Calculus*:
  - Non-Procedural Language
  - Lets users describe what they want, rather than how to compute it

Basic Operators (Relational Algebra)

- Unary Operators
  1. Select
  2. Project
- Binary Operators
  3. Union
  4. Intersection
  5. Set Difference
  6. Cartesian Product

Selection (or Restriction)

- The Selection operation works on a single relation R and defines a relation that contains only those tuples of R that satisfy the specified condition (*predicate*)
- Notation is Greek symbol sigma:

$$\sigma_{PREDICATE}(RELATION)$$

Selection Example

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

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

- The input relation is Staff and the predicate is:

  salary > 10000
- The Selection operation defines a relation containing only those Staff tuples with a salary greater than £10,000. The result of this operation is shown on next slide

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
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

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

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

- More complex predicates can be generated using the logical operators:

  $\wedge$ (AND), $\vee$ (OR) and ~ (NOT)

Projection

- The Projection operation 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.
- Notation:

$$\Pi_{a1, a2, ..., ak}(R)$$

where a_1 ... a_k are attribute names and R is a relation name

Projection (Example)

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

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

- In this example, the Projection operation defines a relation that contains only the designated Staff attributes staffNo, fName, lName, and salary, in the specified order
- The result of this operation is shown on next slide

Projecting the Staff relation over the staffNo, fName, lName, and salary attributes

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
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

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

Set Operations
Union **R** ∪ **S**

- The union of two relations R and S defines a relation that contains all the tuples of R and S both, duplicate tuples being eliminated
- R and S must be union-compatible

Union

- Union-Compatibility
  - Union is possible only if the schemas of the two relations match, that is, if they have the same common attribute(s) having the same domain
- Union is Commutative

$$R \cup S = S \cup R$$

Union Example

Course1

CID	ProgID	Cred_Hrs	CourseTitle
C2345	P1245	3	Operating Sytems
C3456	P1245	4	Database Systems
C4567	P9873	4	Financial Management
C5678	P9873	3	Money & Capital Market

Course2

CID	ProgID	Cred_Hrs	CourseTitle
C4567	P9873	4	Financial Management
C8944	P4567	4	Electronics

Union Example

Course 1 U Course 2

CID	ProgID	Cred_Hrs	CourseTitle
C2345	P1245	3	Operating Sytems
C3456	P1245	4	Database Systems
C4567	P9873	4	Financial Management
C5678	P9873	3	Money & Capital Market
C8944	P4567	4	Electronics

Intersection **R ∩ S**

- The Intersection 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
- Intersection is Commutative

$$R \cap S = S \cap R$$

Intersection (∩) Example

Course 1 ∩ Course 2

CID	ProgID	Cred_Hrs	CourseTitle
C4567	P9873	4	Financial Management

Set Difference **R – S**

- The Set difference operation defines a relation consisting of the tuples that are in relation R, but not in S
- R and S must be union-compatible
- Note that:

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

Difference (-) Example

Course1

CID	ProgID	Cred_Hrs	CourseTitle
C2345	P1245	3	Operating Sytems
C3456	P1245	4	Database Systems
C4567	P9873	4	Financial Management
C5678	P9873	3	Money & Capital Market

Course2

CID	ProgID	Cred_Hrs	CourseTitle
C4567	P9873	4	Financial Management
C8944	P4567	4	Electronics

Difference (-) Example

Course1 – Course2

CID	ProgID	Cred_Hrs	CourseTitle
C2345	P1245	3	Operating Sytems
C3456	P1245	4	Database Systems
C5678	P9873	3	Money & Capital Market

Cartesian Product **R** × **S**

- Sets do not have to be union compatible
- The Cartesian product operation defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S
- Also called “cross product”

Cartesian Product (X) Example

Course X Registration

Course

CID	CourseTitle
C3456	Database Systems
C4567	Financial Management
C5678	Money & Capital Market

Registration

SID	StudName
S101	Ali Tahir
S103	Farah Hasan

Cartesian Product (X) Example

CID	CourseTitle	SID	StudName
C3456	Database Systems	S101	Ali Tahir
C4567	Financial Management	S101	AliTahr
C5678	Money & Capital Market	S101	Ali Tahir
C3456	Database Systems	S103	Farah Hasan
C4567	Financial Management	S103	Farah Hasan
C5678	Money & Capital Market	S103	Farah Hasan

Cartesian Product

- The Cartesian product operation multiplies two relations to define another relation consisting of all possible pairs of tuples from the two relations
- If one relation has *I* tuples and *N* attributes and the other has *J* tuples and *M* attributes, the Cartesian product relation will contain (*I* * *J*) tuples with (*N* + *M*) attributes
- It is possible that the two relations may have attributes with the same name. In this case, the attribute names are prefixed with the relation name to maintain the uniqueness of attribute names within a relation

Cartesian Product Example

- ***List the names and comments of all clients who have viewed a property for rent***
- *Client* relation stores names of clients
- *Viewing* relation stores comments
- To obtain the list of clients and the comments on properties they have viewed, we need to combine these two relations:

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

Client

clientNo	fName	lName	telNo	prefType	maxRent
CR76	John	Kay	0207-774-5632	Flat	425
CR56	Aline	Stewart	0141-848-1825	Flat	350
CR74	Mike	Ritchie	01475-392178	House	750
CR62	Mary	Tregear	01224-196720	Flat	600

X

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-04	too small
CR76	PG4	20-Apr-04	too remote
CR56	PG4	26-May-04	
CR62	PA14	14-May-04	no dining room
CR56	PG36	28-Apr-04	

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

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	

Cartesian Product Example

- Resultant relation (in the previous slide) contains more information while we need tuples where Client.clientNo = Viewing.clientNo

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

Restricted Cartesian product of Client and Viewing relations

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

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

Decomposing Complex Operations

- We can decompose such operations into a series of smaller relational algebra operations and give a name to the results of intermediate expressions.
- We use the assignment operation, denoted by ←, to name the results of a relational algebra operation.
- We could rewrite the previous operation as follows:

TempClient(clientNo, fName, lName) $\leftarrow \Pi_{\text{clientNo, fName, lName}}$(Client)

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

Comment(clientNo, fName, lName, vclientNo, propertyNo, comment) ← TempClient × TempViewing

Result $\leftarrow \sigma_{\text{clientNo = vclientNo}}$(Comment)

Join Operations

- Typically, we want only combinations of the Cartesian product that satisfy certain conditions
- Hence we use a **Join operation** instead of the Cartesian product operation
- Combines two relations to form a new relation
- One of the essential operations in the relational algebra

Types of Join Operations

1. Theta join
2. Equijoin (a particular type of Theta join)
3. Natural join
4. Outer join
5. Semijoin

Theta join (θ-join)

- 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: <, ≤, >, ≥, =, ≠

$$R \bowtie_F S$$

Theta join (θ-join)

- We can rewrite the Theta join in terms of the basic Selection and Cartesian product operations:

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

- The degree of a Theta join is the sum of the degrees of the operand relations R and S

Equijoin Operation

- In the case where the predicate *F* contains only **equality (=)**, the term **Equijoin** is used instead

Example:

- *List the names and comments of all clients who have viewed a property for rent*
- In the previous slides, we used the Cartesian product and Selection operations to obtain this list.
- However, the same result is obtained using the Equijoin operation:

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

Equijoin

- Rows are joined on the basis of values of a common attribute between the two relations
- Rows having the same value in the common attribute are joined
- Common attributes appear twice in the output
- Common attribute with the same name is qualified with the relation name in the output

Equijoin Example

FACULTY

facId	facName	dept	salary	rank
F2345	Usman	CSE	21000	lecturer
F3456	Tahir	CSE	23000	Asso Prof
F4567	Ayesha	ENG	27000	Asso Prof
F5678	Samad	MNG	32000	Professor

COURSE

crCode	crTitle	facId
C3456	Database Systems	F2345
C4567	Financial Management	
C5678	Money & Capital Market	F4567
C3425	Introduction to Accounting	F2345

FACULTY $\bowtie$ FACULTY.facId = COURSE.facId COURSE

Equijoin Example

FACULTY ⋈ FACULTY.facId = COURSE.facId COURSE

FACULTY.facId	facName	dept	salary	Rank	crCode	crTitle	COURSE.facId
F2345	Usman	CSE	21000	lecturer	C3456	Database Systems	F2345
F4567	Ayesha	ENG	27000	Asso Prof	C5678	Money & Capital Market	F4567
F2345	Usman	CSE	21000	lecturer	C3425	Introduction to Accounting	F2345

Natural Join $R \bowtie S$

- Also called simply the join
- The most general form of join
- The Natural join is 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

Natural Join Example

FACULTY

facId	facName	dept	salary	rank
F2345	Usman	CSE	21000	lecturer
F3456	Tahir	CSE	23000	Asso Prof
F4567	Ayesha	ENG	27000	Asso Prof
F5678	Samad	MNG	32000	Professor

COURSE

crCode	crTitle	facId
C3456	Database Systems	F2345
C4567	Financial Management	
C5678	Money & Capital Market	F4567
C3425	Introduction to Accounting	F2345

FACULTY ⋈ COURSE

Natural Join Example

FACULTY ⋈ COURSE

facId	facName	dept	salary	Rank	crCode	crTitle
F2345	Usman	CSE	21000	lecturer	C3456	Database Systems
F4567	Ayesha	ENG	27000	Asso Prof	C5678	Money & Capital Market
F2345	Usman	CSE	21000	lecturer	C3425	Introduction to Accounting

Outer Joins

- Often in joining two relations, a tuple in one relation does not have a matching tuple in the other relation; in other words, there is no matching value in the join attributes
- We may want tuples from one of the relations to appear in the result even when there are no matching values in the other relation. This may be accomplished using the Outer join.
- Types:
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join

Left Outer Join **R ⟕ S**

- The Left Outer Join is a join in which tuples from R that do not have matching values in the common attributes of S are also included in the result relation. Missing values in the second relation are set to null
- The advantage of an Outer join is that information is preserved, that is, the Outer join preserves tuples that would have been lost by other types of join

Left Outer Join Example

FACULTY

facId	facName	dept	salary	rank
F2345	Usman	CSE	21000	lecturer
F3456	Tahir	CSE	23000	Asso Prof
F4567	Ayesha	ENG	27000	Asso Prof
F5678	Samad	MNG	32000	Professor

COURSE

crCode	crTitle	facId
C3456	Database Systems	F2345
C4567	Financial Management	
C5678	Money & Capital Market	F4567
C3425	Introduction to Accounting	F2345

facId	facName	dept	salary	rank	crCode	crTitle	facId
F2345	Usman	CSE	21000	lecturer	C3456	Database Systems	F2345
F2345	Usman	CSE	21000	lecturer	C3425	Intro. To Accounting	F2345
F3456	Tahir	CSE	23000	Asso Prof			
F4567	Ayesha	ENG	27000	Asso Prof	C5678	Money & Capital Market	F4567
F5678	Samad	MNG	32000	Professor			

Right Outer Join **R⟖S**

Same as the left, but keep tuples from the "right" relation

Right Outer Join Example

FACULTY

facId	facName	dept	salary	rank
F2345	Usman	CSE	21000	lecturer
F3456	Tahir	CSE	23000	Asso Prof
F4567	Ayesha	ENG	27000	Asso Prof
F5678	Samad	MNG	32000	Professor

COURSE

crCode	crTitle	facId
C3456	Database Systems	F2345
C4567	Financial Management	
C5678	Money & Capital Market	F4567
C3425	Introduction to Accounting	F2345

facId	facName	dept	salary	rank	crCode	crTitle	facId
F2345	Usman	CSE	21000	lecturer	C3456	Database Systems	F2345
					C4567	Financial Management	
F4567	Ayesha	ENG	27000	Asso Prof	C5678	Money & Capital Market	F4567
F2345	Usman	CSE	21000	lecturer	C3425	Intro. To Accounting	F2345

Full Outer Join

All matching tuples plus all non-matching tuples from both relations

Full Outer Join Example

FACULTY

facId	facName	dept	salary	rank
F2345	Usman	CSE	21000	lecturer
F3456	Tahir	CSE	23000	Asso Prof
F4567	Ayesha	ENG	27000	Asso Prof
F5678	Samad	MNG	32000	Professor

COURSE

crCode	crTitle	facId
C3456	Database Systems	F2345
C4567	Financial Management	
C5678	Money & Capital Market	F4567
C3425	Introduction to Accounting	F2345

facId	facName	dept	salary	rank	crCode	crTitle	facId
F2345	Usman	CSE	21000	lecturer	C3456	Database Systems	F2345
F2345	Usman	CSE	21000	lecturer	C3425	Intro. To Accounting	F2345
F4567	Ayesha	ENG	27000	Asso Prof	C5678	Money & Capital Market	F4567
F3456	Tahir	CSE	23000	Asso Prof			
F5678	Samad	MNG	32000	Professor			
					C4567	Financial Management	

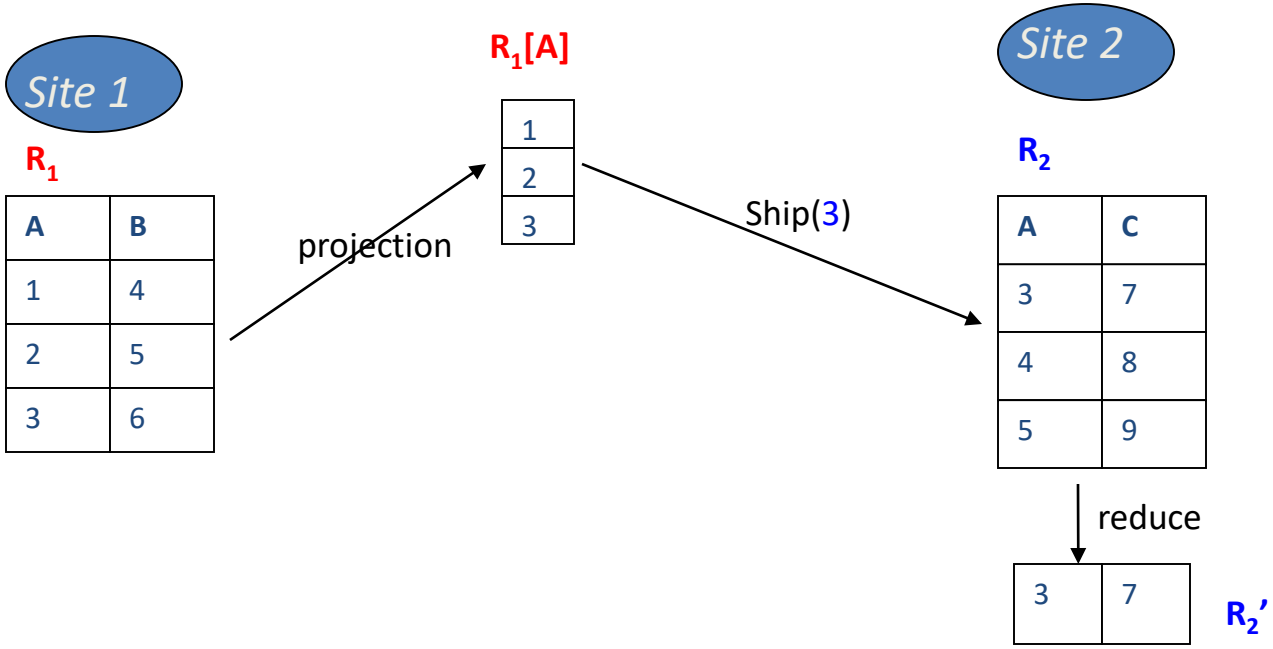
Semijoin $\mathbf{R} \triangleright_F \mathbf{S}$

- A semijoin from R_i to R_j on attribute A can be

  denoted as $R_j \triangleright R_i$ It is used to reduce the data

  transmission cost
- Computing steps:
  1) Project R_i on attribute A ($\Pi_A R_i$) and *ship this projection* (a semijoin projection) from the site of R_i to the site of R_j ;
  2) Reduce R_j to R_j' by eliminating tuples where attribute A are not matching any value in $R_i[A]$

Semijoin Example



Division Operation **R ÷ S**

- Suppose C is the set of attributes of R that are not attributes of S
- The Division operation defines a relation over the attributes C that consists of the set of tuples from R that match the **combination** of **every** tuple in S
- We can express the Division operation in terms of the basic operations:

$$T_1 \leftarrow \Pi_C(R)$$
$$T_2 \leftarrow \Pi_C((T1 \times S) - R)$$
$$T \leftarrow T_1 - T_2$$

Division Operation **V ÷ W** Example

Let C is the set of attributes of V that are not attributes of W.

Hence C = attribute A

V

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

W

B
1
2

$T_1 \leftarrow \Pi_A(V)$

A
a
a
b
b
c

$T_1 \times W$

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

$(T_1 \times W) - V$

A	B
c	2

$T_2 \leftarrow \Pi_A((T_1 \times W) - V)$

A
c

$T \leftarrow T_1 - T_2$

A
a
b

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

$$T_2 \leftarrow \Pi_C((T1 \times S) - R)$$

$$T \leftarrow T_1 - T_2$$

Examples of Division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

sno
s1
s2
s3
s4

A/B1

pno
p2
p4

B2

sno
s1
s4

A/B2

pno
p1
p2
p4

B3

sno
s1

A/B3

More Examples of Relational Algebra Queries

Find names of sailors who've reserved boat #103

Solution1: $Temp1 \leftarrow \sigma_{bid=103}(Reserves)$

$Temp2 \leftarrow (Temp1 \bowtie Sailors)$

$\pi_{sname}(Temp2)$

Solution2: $\pi_{sname}(\sigma_{bid=103}(Reserves \bowtie \text{Sailors}))$

Solution3: $\pi_{sname}((\sigma_{bid=103}(Reserves)) \bowtie \text{Sailors})$

Sailors

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reserves

sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/5/98

Boats

bid	bname	color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	Green
104	Marine	Red

Find names of sailors who've reserved a red boat

- Information about boat color only available in Boats; so need an extra join:

$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie Reserves \bowtie Sailors)$$

  - A more efficient solution:

$$\pi_{sname}(\pi_{sid}((\pi_{bid}\sigma_{color='red'} Boats) \bowtie Res) \bowtie Sailors)$$

A query optimizer can find this, given the first solution!

Summary

- Relational algebra is more operational; useful as internal representation for query evaluation plans
- Several ways of expressing a given query; a query optimizer should choose the most efficient version

Relational Calculus

Relational Calculus

- Relational calculus is a non-procedural formal data manipulation language
- Simply specifies what data should be retrieved

Relational Calculus

- There is no description of how to evaluate a query
- A Relational Calculus query specifies *what* is to be retrieved rather than *how* to retrieve it
- Relational Calculus takes its name from a branch of symbolic logic called **predicate calculus**

Relational Calculus

- In **predicate calculus**, a **predicate** is a truth-valued **function** with arguments
- When we substitute values for the arguments, the **function** yields an expression, called a **proposition**, which can be either **true or false**
- For example, the sentences:
  - 'John White is a member of staff' and
  - 'John White earns more than Ann Beech'
- are both **propositions**, since we can determine whether they are true or false.
- In the first case, we have a **function**, **'is a member of staff'**, with one argument **(John White)**
- In the second case, we have a function, **'earns more than'**, with two arguments **(John White and Ann Beech)**

Relational Calculus

- If a **predicate** contains a variable, as in **'*x* is a member of staff'**, there must be an associated **range for *x***
- When we substitute some values of this range for *x*, the proposition may be true; for other values, it may be false
- For example, **if the range is the set of all people** and we replace ***x* by John White**, the proposition 'John White is a member of staff' is true
- If we replace *x* by the name of a person who is not a member of staff, the proposition is false

Relational Calculus

- Comes in two forms
  1. Tuple Relational Calculus
  2. Domain Relation Calculus

Tuple Relational Calculus

- Here we are interested in **finding tuples** for which a **predicate is True**
- The calculus is based on the use of **tuple variables**
- A tuple variable is a variable that **'ranges over'** a named relation: that is, a variable whose only permitted values are tuples of the relation
- For example, to specify the range of a **tuple variable *S*** over the Staff relation, we write:

$$\text{Staff}(S)$$

- To express the query 'Find the set of all tuples *S* such that *F*(*S*) is true', we can write:

$$\{S \mid F(S)\}$$

Tuple Relational Calculus

- *F* is called a **formula** (well-formed formula, or **wff** in mathematical logic)
- For example, to express the query 'Show all attributes of staff earning more than £10,000', we can write:

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

- S.salary means the value of the salary attribute for the tuple variable *S*
- To retrieve a particular attribute, such as salary, we would write:

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

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

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
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Expressions and formulae

An expression in the tuple relational calculus has the following general form:

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

- where $S_1, S_2, \ldots, S_n \ldots, S_m$ are tuple variables
- each a_i is an attribute of the relation over which S_i ranges
- F is a formula

Expressions and formulae

A (well-formed) formula is made out of one or more atoms, where an atom has one of the following forms:

- $R(S_i)$

  where S_i is a tuple variable and R is a relation
- $S_i.a_1 \ \theta \ S_j.a_2$

  where S_i and S_j are tuple variables

  a_1 is an attribute of the relation over which S_i ranges

  a_2 is an attribute of the relation over which S_j ranges

  θ is one of the comparison operators ($<$, $>$, $\geq$, $\leq$, $\neq$, $=$)
- $S_i.a_1 \ \theta \ c$

  where S_i is a tuple variable

  a_1 is an attribute of the relation over which S_i ranges

  c is a constant from the domain of attribute a_1

  θ is one of the comparison operators

The Existential and Universal Quantifiers

- There are two **quantifiers** we can use with formulae to tell how many instances the predicate applies to
- The **existential quantifier** ∃ (‘there exists’) is used in formulae that must be true for at least one instance, such as:

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

- This 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’

The Universal Quantifier

- The **universal quantifier** ∀ (‘for all’) is used in statements about every instance, such as:

  (∀B) (B.city ≠ ‘Paris’)

- This means, ‘For all Branch tuples, the city is not Paris’

  OR ~(∃B) (B.city = ‘Paris’)

which means, ‘There is not a single branch with city equals to Paris’

Bound Variables & Free Variables

- Tuple variables that are qualified by ∀ or ∃ are called **bound variables**, otherwise the tuple variables are called **free variables**
- The only **free variables** in a relational calculus expression should be those on the left side of the bar (|)
- For example, in the following query:

{S.fName, S.lName | Staff(S) ∧ (∃B) (Branch(B) ∧ (B.branchNo = S.branchNo) ∧ B.city = ‘London’)}

- S is the only free variable and S is then bound to each tuple of Staff relation

Tuple Relational Calculus Examples

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

{S.fName, S.lName | Staff(S) ∧ S.position = ‘Manager’ ∧ S.salary > 25000}

- *(b) List the **staff** who manage properties for rent in Glasgow*

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

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
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Safety of Expressions

- It is possible for a calculus expression to generate an infinite set. For example

  $$\{S \mid \sim Staff(S)\}$$

- Means the set of all tuples that are not in the Staff relation
- Such an expression is said to be **unsafe**
- To avoid this, we have to add a restriction that all values that appear in the result must be values in the *domain* of the relation(s) appearing in the expression

Domain Relational Calculus

- We also use variables but in this case the variables take their values from ***domains*** of attributes rather than tuples of relations
- General form of domain relational calculus expression:

$$\{d_1, d_2, \ldots, d_n \mid F(d_1, d_2, \ldots, d_m)\}\ m \geq n$$

- where $d_1, d_2, \ldots, d_n$ represent domain variables

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
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Domain Relational Calculus

- *Find the first and last names of all managers who earn more than £25,000*
- {fN, lN | (∃sN, posn, gender, dob, sal, bN)

  (Staff(sN, fN, lN, posn, gender, dob, sal, bN)

  ∧ posn = ‘Manager’ ∧ sal > 25000)}
- Here, each attribute is given a (variable) name

Domain Relational Calculus

- When the **domain relational calculus** is restricted to **safe** expressions, it is equivalent to the **tuple relational calculus** restricted to **safe** expressions, which in turn is equivalent to the **relational algebra**
- This means that for every relational algebra expression there is an equivalent expression in the relational calculus (tuple or domain relational calculus), and for every relational calculus expression there is an equivalent relational algebra expression

Chapter Summary

- Relational algebra and relational calculus are equivalent to one another: for every expression in the algebra, there is an equivalent expression in the calculus (and vice versa).
- Relational calculus is used to measure the **selective power of relational languages**
- A **language** (e.g. SQL) that can be used to produce any **relation** which can be derived using the **relational calculus** is said to be **relationally complete**
- Most relational query languages are **relationally complete** but have more expressive power than the relational algebra or relational calculus because of additional operations such as **calculated summary and ordering functions**