Databases

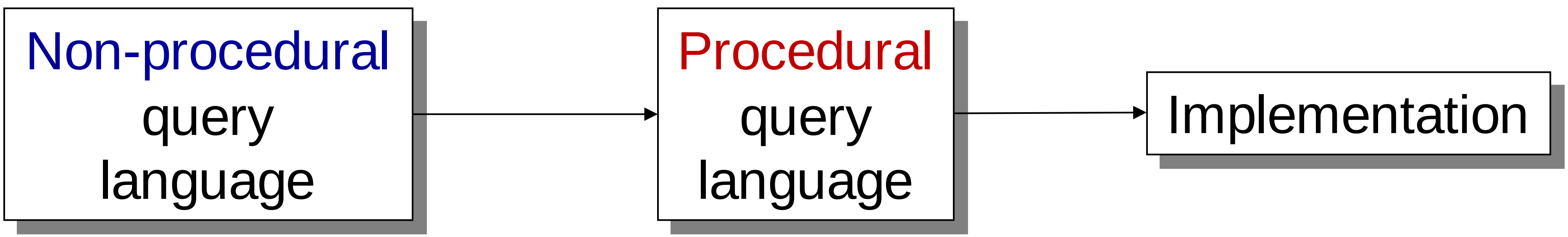
Relational Calculus & Relational Algebra

Dr Konrad Dabrowski
konrad.dabrowski@durham.ac.uk

Online Office Hour:
Mondays 13:30–14:30
See Duo for the Zoom link

Relational Calculus & Algebra

- Relational calculus: formal *definition* of a new relation from existing relations in the DB
- Relational algebra: *how* to build a new relation from existing relations in the DB
- Their place in the big picture:



SQL, relational calculus

relational algebra

specifies which data are to be retrieved

specifies how to retrieve the required data

Relational Calculus & Algebra

- Relational Calculus:
  - relations are considered as sets of elements (attribute values)
  - the new relation is defined from the old one(s) using a set theoretic expression
- Relational Algebra:
  - based on mathematical relations (tables)
  - a theoretical language with operations on *one (or more)* input relations (tables) to define *one* new (output) relation
  - the input relation(s) remain unchanged
- Relational Algebra and Calculus:
  - based on logic; equivalent languages (same expressive power)
  - for every algebra expression ↔ an equivalent calculus expression

Relational Calculus & Algebra

- Both are formal and non-user-friendly languages
  - used as the basis for higher-level DML such as SQL

    (i.e. the queries of a new language can be described using relational algebra/calculus expressions)

- A query language is relationally complete if:
  - it can be used to produce any relation that can be derived using relational algebra/calculus expressions

- Most modern query languages (like SQL):
  - are relationally complete (but also more than that!)
  - have additional operations *(summing, grouping, ordering, …)*
  - more expressive power than relational algebra
  - but still: not expected to be "Turing complete":
    - i.e. not as powerful as other programming languages
    - just designed to support easy & efficient access to large data sets

Relational Calculus

In *first-order logic* (or *'predicate calculus'*):

- predicate: truth-valued *function* (true/false) with arguments
- proposition: the *expression* obtained when we substitute values to the arguments of a predicate (can be true/false)

- Let $P(x)$ and $Q(x)$ be two predicates with argument x. Then:
  - the 'set of all x such that both $P(x)$ and $Q(x)$ are true' is:

$$\{x \mid P(x) \wedge Q(x)\}$$

  - the 'set of all x such that $P(x)$ or $Q(x)$ is true' is:

$$\{x \mid P(x) \vee Q(x)\}$$

  - the 'set of all x such that $P(x)$ is not true' is:

$$\{x \mid \sim P(x)\}$$

Tuple Relational Calculus

- variables $\longleftrightarrow$ tuples of a relation
- Aim: to find tuples for which some predicate(s) is (are) true
- To specify that a tuple S belongs to the relation Staff, we use the predicate: $\mathrm{Staff}(S)$

Examples:

- 'all *tuples* S of the relation Staff that have salary > 10000':

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

- 'the *salaries* of all members of staff which earn > 10000':

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

Domain relational calculus (another type of calculus):

- variables take values from domains of attributes of a relation (instead of tuples)

Tuple Relational Calculus

- use of <u>quantifiers</u> while building predicates:
  - existential quantifier: $\exists$ ('there exists')
  - universal quantifier: $\forall$ ('for all')
- tuple variables that are:
  - quantified by $\exists$ or $\forall$ $\longrightarrow$ 'bound variables'
  - not quantified by $\exists$ or $\forall$ $\longrightarrow$ 'free variables'

Example:

- 'the names of all staff members who work in a branch in London':

$$\{S.name \mid \text{Staff}(S) \wedge (\exists B)(\text{Branch}(B) \wedge (B.branchNo = S.branchNo) \wedge (B.city = \text{'London'}))\}$$

S : free variable

B : bound variable

Relational Algebra

- Both input and output are relations

  ⇒ the output can become input to another operation

  ⇒ expressions can be nested

  – this property is called "closure"

- Relations are closed under Relational Algebra:

  – in the same sense as:

    "numbers are closed under arithmetic expressions"

- In Relational Algebra:

  – all involved tuples from the input relation(s) are manipulated in one statement (with no loops)

Relational Algebra

- Six basic operations:
  - Selection (σ)
  - Projection (π)
  - Rename (ϱ)
  - Union (∪)
  - Set Difference (–)
  - Cartesian product (×)

Unary operations

Binary operations

- Several derived operations
  (they can be expressed using the basic operations):
  - Intersection (∩)
  - Division (÷)
  - Join (natural join, equi-join, theta join, outer join, semi-join)

Selection

- **$\sigma_{predicate}$ (R)**
  - unary operation, i.e. it works on a single relation R
  - outputs a subset of the relation R that contains only the tuples (rows) that satisfy the specified condition (*predicate*)
  - i.e. it returns a "horizontal slice" of R

Example: **List all staff whose salary is greater than 12000**

Staff

StaffNo	FName	Sname	Position	Salary	Branch
SL41	Julie	Lee	Assistant	9000	B005
SL21	John	White	Manager	30000	B005
SA9	Mary	Howe	Assistant	11000	B007
SG37	Ann	Beech	Supervisor	18000	B005
SL14	David	Ford	Assistant	8000	B007
SG5	Sue	Brand	Manager	25000	B006

$\sigma_{salary > 12000}$ (Staff)

StaffNo	FName	Sname	Position	Salary	Branch
SL21	John	White	Manager	30000	B005
SG37	Ann	Beech	Supervisor	18000	B005
SG5	Sue	Brand	Manager	25000	B006

Projection

- $\Pi_{\text{col-1}, \ldots, \text{col-n}}(R)$
  – unary operation, i.e. it works on a single relation R
  – outputs a subset of the relation R that contains only the specified attributes (columns) with names col-1, …, col-n and also eliminates duplicates
  – i.e. it returns a "vertical slice" of R (by removing non-matching attributes)

Produce a list of salaries for all staff, showing only staffNo, fName, lName, and salary

Staff

StaffNo	FName	Sname	Position	Salary	Branch
SL41	Julie	Lee	Assistant	9000	B005
SL21	John	White	Manager	30000	B005
SA9	Mary	Howe	Assistant	11000	B007
SG37	Ann	Beech	Supervisor	18000	B005
SL14	David	Ford	Assistant	8000	B007
SG5	Sue	Brand	Manager	25000	B006

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

StaffNo	FName	Sname	Salary
SL41	Julie	Lee	9000
SL21	John	White	30000
SA9	Mary	Howe	11000
SG37	Ann	Beech	18000
SL14	David	Ford	8000
SG5	Sue	Brand	25000

Projection

- We can combine Selection and Projection:

List the last names and salaries of all staff with salary greater than 15000

Staff

StaffNo	FName	lName	Position	Salary	Branch
SL41	Julie	Lee	Assistant	9000	B005
SL21	John	White	Manager	30000	B005
SA9	Mary	Howe	Assistant	11000	B007
SG37	Ann	Beech	Supervisor	18000	B005
SL14	David	Ford	Assistant	8000	B007
SG5	Sue	Brand	Manager	25000	B006

$$\Pi_{\mathbf{lName,salary}}(\sigma_{\mathbf{salary\ >\ 15000}}\ \mathbf{(Staff)})$$

lName	Salary
White	30000
Beech	18000
Brand	25000

- Why do we ever need to remove duplicates?

List the branches of all staff with salary greater than 15000

$$\Pi_{\mathbf{Branch}}(\sigma_{\mathbf{salary\ >\ 15000}}\ \mathbf{(Staff)})$$

Branch
B005
B006

without removing duplicates we would obtain:

Branch
B005
B005
B006

Union

- **R ∪ S**
  - binary operation, i.e. it works on two relations R and S
  - outputs a new relation having all tuples of R, or S, or both R and S, and also eliminates duplicate tuples
- That is:
  - it combines the rows from both tables, removing any redundant (common) row in the process
  - if R has I tuples and S has J tuples, the output relation will have at most $I + J$ tuples

List all cities where there is either a branch office or a property for rent (or both)

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

city
London
Aberdeen
Glasgow
Bristol

Set Difference

- **R – S**
  - binary operation, i.e. it works on two relations R and S
  - outputs a new relation having all tuples that exist in R but not in S
- That is:
  - it removes from R any common rows that appear in both tables R and S
  - if R has I tuples and S has J tuples, the output relation will have at least $I - J$ tuples
  - similarly, $S - R$ removes from S its common rows with R

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

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

city
Bristol

Union compatibility

- To compute $R \cup S$ and $R - S$:
  - the schemas of the relations R and S must match, i.e.
    - R and S must have the same number of attributes
    - every pair of corresponding attributes must have the same domain
  - then R and S are called union-compatible

- What if one of the relations has extra attributes?
  - usual trick: use projection to create union-compatible relations:

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

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

Intersection

- **$R \cap S$**
  - binary operation, i.e. it works on two relations R and S
  - the output relation has all tuples existing in both R and S
- That is:
  - it removes from R any rows that appear only in R
  - equivalently: it removes from S any rows appearing only in S
  - if R has I tuples and S has J tuples, the output relation will have at most $\min\{I, J\}$ tuples
- To compute $R \cap S$:
  - the relations R and S must be again union-compatible
- Intersection is a derived (i.e. not basic) operation:

$$\mathbf{R \cap S = R - (R - S)}$$

Examples

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

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

city
Aberdeen
London
Glasgow

R

StaffNo	Sname	Salary	Branch
SL14	Ford	8000	B007
SG5	Brand	25000	B006
SA9	Howe	11000	B007

S

StaffNo	Sname	Salary	Branch
SL41	Lee	9000	B005
SL21	White	30000	B005
SA9	Howe	11000	B007
SG37	Beech	18000	B005

R ∪ S

StaffNo	Sname	Salary	Branch
SL41	Lee	9000	B005
SL21	White	30000	B005
SA9	Howe	11000	B007
SG37	Beech	18000	B005
SL14	Ford	8000	B007
SG5	Brand	25000	B006

R – S

StaffNo	Sname	Salary	Branch
SL14	Ford	8000	B007
SG5	Brand	25000	B006

R ∩ S

StaffNo	Sname	Salary	Branch
SA9	Howe	11000	B007

Cartesian Product

- **R x S**
  - binary operation, i.e. it works on two relations R and S
  - outputs a new relation that is a concatenation of every tuple from R with every tuple from S
  - no further "compatibility" assumptions on the relations
  - recall: the ordering of the tuples does not matter!
- That is:
  - it "multiplies" the relations R and S
  - if R has I tuples, N attributes and S has J tuples, M attributes then $R \times S$ has $(I * J)$ tuples and $(N + M)$ attributes
- If R and S have attributes with the same name:
  - the attribute names are prefixed with the relation name
  - e.g. $R.name$ and $S.name$

Cartesian Product

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

- To obtain the list of clients and comments of properties they viewed:
  - we need to combine the relations *Client* and *Viewing*:

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

prefixed attributes →

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	

However:

- more information than required!
- in many rows we have different values for *clientNo*

$\Rightarrow$ we need to eliminate them!

Cartesian Product

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

- To obtain the list of clients and comments of properties they viewed:
  - we need to combine the relations *Client* and *Viewing* :

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

⟹ use selection operation to extract those tuples where Client.clientNo = Viewing.clientNo:

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

selection predicate

equal *clientNo* values

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

Rename

- A Relational Algebra operations can be very complex
  - we decompose it into a series of smaller operations
  - we give names to the intermediate expressions (to reuse them)
  - for this we can iteratively use the assignment operation "←"

- A simple (but very useful!) alternative:
  - the rename operation $\varrho_X(E)$ returns E renamed as X
  - moreover: $\varrho_{X(A_1, A_2, \ldots, A_n)}(E)$ returns E renamed as X, where the attributes of X are defined as $A_1, A_2, \ldots, A_n$

- Especially useful when, for example:
  - we want to compute a join of a relation with itself (i.e. we create a new copy of the relation with a different name)

Rename – Example (1)

Family

Parent	Child
Mary	Thomas
Peter	Mary
Sarah	Nick
Nick	Thomas
Andrew	Nick
Nick	Helen
Kate	Mary

Given a table with information about parent/child pairs, find all grandparents of Thomas

- **We need to find the "parents of parents" of Thomas**

$\Longrightarrow$ **we need two copies of the Family table:**

Copy "A"

Parent	Child
Mary	Thomas
Peter	Mary
Sarah	Nick
Nick	Thomas
Andrew	Nick
Nick	Helen
Kate	Mary

Copy "B"

Parent	Child
Mary	Thomas
Peter	Mary
Sarah	Nick
Nick	Thomas
Andrew	Nick
Nick	Helen
Kate	Mary

$$\Pi_{A.Parent}\left(\sigma_{(A.Child\ =\ B.Parent)\ \wedge\ B.Child\ =\ \text{'Thomas'}}\left(\rho_A(Family) \times \rho_B(Family)\right)\right)$$

Rename – Example (2)

- Account:

number	branch	balance
A_101	Durham	1000
A_102	Newcastle	2000
A_103	London	3000

- Query: find the largest account balance in the bank.
  - Balances that are *not* the largest:

$$T \leftarrow \Pi_{\text{account.balance}}(\sigma_{\text{account.balance} < \text{d.balance}}(\text{account} \times \rho_{\text{d}}(account)))$$

    - Result: {1000, 2000}

  - The *largest* account balance: $\Pi_{\text{account.balance}}(account) - T$
    - Result: {3000}

Division

- **R ÷ S**
  - binary operation, i.e. it works on two relations R and S
  - a particular type of query that appears often in applications

- Notation:
  - let R have the set of attributes A
  - let S have the set of attributes B, where $B \subseteq A$
  - define $C = A - B$ (i.e. the attributes of R that are not in S)

- The division operator R ÷ S :
  - outputs a relation over the attributes C that consists of the tuples from R that match every tuple in S

Division – Example

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

$\Longrightarrow$

RESULT

clientNo
CR56

- Division R ÷ S is a derived (i.e. not basic) operation:
  - compute all *C*-tuples of *R* that are not "disqualified" by a tuple in *S*
  - a *C*-tuple of *R* is "disqualified", if by attaching to it a tuple of *S*, we obtain a tuple that is not in *R*
  - disqualified *C*-tuples of *R*: $\Pi_C((\Pi_C(R) \times S) - R)$
  - tuples of R ÷ S : $\Pi_C(R)$ – disqualified tuples

Join: a derivative of Cartesian product

- The combination of Cartesian product and Selection can be reduced to a single operation, called a Join
- A join is equivalent to:
  - build the Cartesian product of the two operand relations
  - perform a Selection (using the join predicate F)

- Notation:
  - $R \bowtie_F S = \sigma_F(R \times S)$, where F is a predicate, e.g.:

$$\text{Client} \bowtie_{(\text{Client.clientNo} = \text{Viewing.clientNo})} \text{Viewing} =$$

$$\sigma_{\text{Client.clientNo} = \text{Viewing.clientNo}}(\ \text{Client} \times \text{Viewing}\ )\ )$$

  - if F contains only "=", then this is an Equijoin

Natural Join

- $A_1, A_2, \ldots . A_k$: common attributes of relations *R* and *S*
- $R \bowtie S = \Pi_{C_1,\ldots,C_x}(\sigma_{R.A_1=S.A_1,\ldots,R.A_k=S.A_k}(R \times S)),$

  where $C_1, \ldots . C_x$ are the attributes of R×S without the duplicates
- it is an Equijoin of the relations *R* and *S* over all their <u>attributes</u> that have the <u>same name</u>, without duplications

- Steps:
  - $R \times S$
  - For each attribute with the same name *A* in both *R* and *S*, select tuples where *R.A=S.A* (from *R×S*)
  - Remove one of the duplicate columns corresponding to the above pairs of attributes

Natural Join (example)

- Relations *R* and *S* :

R :

A	B	C	D
x	1	x	a
y	2	z	a
z	4	y	b
x	1	z	a
w	2	y	b

S :

B	D	E
1	a	x
3	a	y
1	a	z
2	b	w
3	b	t

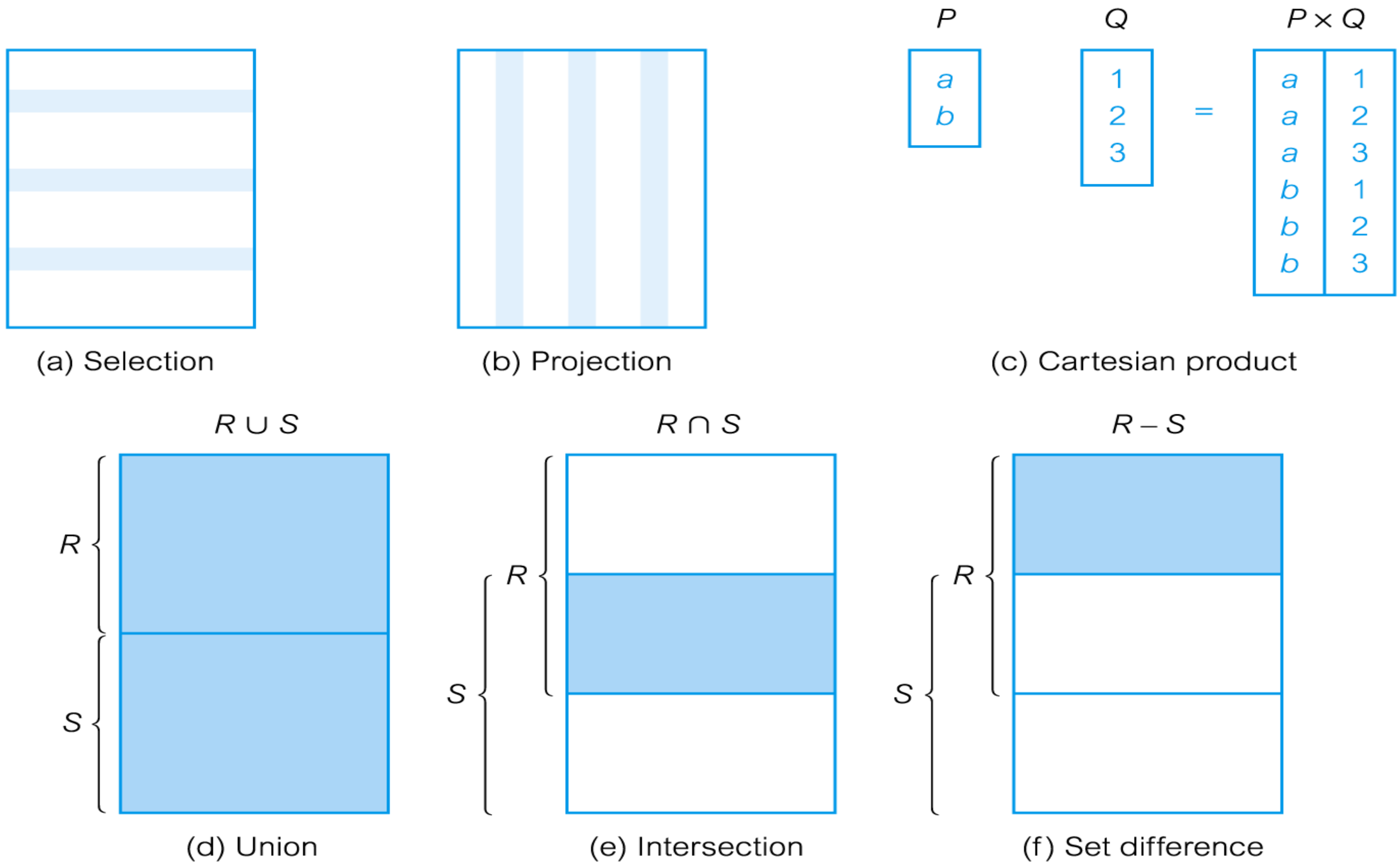
$R \bowtie S$:

A	B	C	D	E
x	1	x	a	x
x	1	x	a	z
x	1	z	a	x
x	1	z	a	z
w	2	y	b	w

Natural Join (exercise)

- Given the schemas R(A, B, C, D) and S(A, C, E), what is the schema of R ⋈ S ?
  - Answer: (A, B, C, D, E)
- Given R(A, B, C) and S(D, E), what is R ⋈ S ?
  - Answer: $R \times S$
- Given R(A, B) and S(A, B), what is R ⋈ S ?
  - Answer: $R \cap S$

Summary: Relational Algebra operations



Try these for exercise

Describe the relation(s) that would be produced by the following relational algebra operations:

$$\Pi_{\text{hotelNo}} (\sigma_{\text{price} > 50} (\text{Room}))$$

This will produce a relation with a single attribute (hotelNo) giving the numbers of those hotels with a room price greater than £50.

$$\sigma_{\text{Hotel.hotelNo} = \text{Room.hotelNo}} (\text{Hotel} \times \text{Room})$$

This will produce a join of the Hotel and Room relations containing all the attributes of both Hotel and Room (there will be two copies of the hotelNo attribute). Essentially this will produce a relation containing all rooms at all hotels

Summary of the Lecture

- Procedural and non-procedural languages
- Relational Calculus (tuple / domain calculus)
- Relational algebra:
  - Selection
  - Projection
  - Union
  - Set Difference
  - Cartesian product
  - Rename
  - Intersection
  - Join

Additional Slides

Outer Join

- $R \mathbin{⟕} S$ (left outer join):
- An extension of the Natural Join operation that avoids loss of information.

- Computes the Natural Join $R \bowtie S$ and then:
  - adds to the result tuples from relation R that do not match tuples in relation S

- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist

Outer Join (example)

- Relation *Loan*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

- Relation *Borrower*

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

Outer Join (example)

- Natural Join
  - *Loan* $\bowtie$ *Borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

- Left Outer Join
  - *Loan* ⟕ *Borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

Outer Join (example)

- Right Outer Join
  - *Loan* ⟖ *Borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

- Full Outer Join
  - *Loan* ⟗ *Borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Semijoin

- $R \triangleright_F S = \Pi_A(R \bowtie_F S)$, where *A* is the set of attributes in *R*:
  - defines a relation which contains the tuples of *R* that participate in the join of *R* and *S* (according to the predicate *F*).

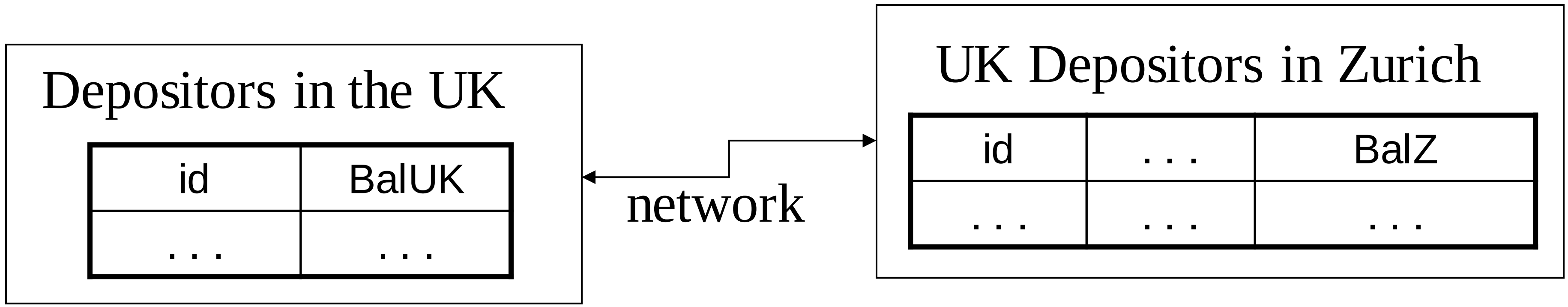
- Example:
- 'List complete details of all staff who work at the branch in Paris'.

$$\text{Staff} \triangleright_{\text{Staff.branchNo}=\text{Branch.branchNo}} (\sigma_{\text{city}=\text{Paris}}(\text{Branch}))$$

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

(the Natural Join: would have additionally the details of the relation Branch;
the Semijoin: has only the attributes of the relation Staff)

Semijoins in Distributed Databases



- <u>Our aim:</u> to find all UK citizens with total balance (in the UK and Zurich) greater than 100M (i.e. 100,000,000)

$$\text{DepUK} \bowtie_{\text{DepUK.id=DepZ.id, BalUK+BalZ>100M}} (\text{DepZ})$$

- answer in distributed DB:

$T = \text{DepZ}$

$R = \text{DepUK} \rhd_F T$

$\text{Answer} = R \bowtie \text{DepZ}$

where F: $\text{BalUK} + \text{BalZ} > 100\text{M}$

Aggregation Operations

- Recall:
  - modern query languages (e.g. SQL) have more operations than relational algebra (e.g. *summing*, *grouping*, *ordering*, …)

$\Longrightarrow$ additional operations were proposed for relational algebra

  - to describe formally higher-level query languages
  - these operations cannot be expressed using the basic operations

---

Example: aggregate operations $\mathfrak{I}_{AL}(R)$

- unary operation, i.e. it works on a single relation R
- it applies the aggregate-function-list AL to R
- AL contains several pairs of the form (<***aggr_function***>, <***attribute***>)
- outputs a relation over the aggregate list

Aggregation Operations

- The main aggregate functions:
  - COUNT: the number of values in the associated attribute
  - SUM: the sum of the values in the associated attribute
  - AVG: the average of the values in the associated attribute
  - MIN / MAX: the smallest/largest value in the associated attribute

---

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

$$\mathfrak{I}_{\text{COUNT propertyNo}} (\sigma_{rent > 350} (\text{PropertyForRent}))$$

Find the minimum, maximum, and average staff salary

$$\mathfrak{I}_{\text{MIN salary, MAX salary, AVG salary}} (\text{Staff})$$

Attribute-function-list AL