CSCU9B3

Database Principles and Applications

Relational Algebra

Relational Algebra

- The *relational algebra* is a theoretical language
  - It is the theoretical basis of query languages such as SQL.
  - It contains operators that work on one or more relations.
  - These operators give us the means to construct new relations from given ones.
  - It is similar in some ways to ordinary arithmetic (often called algebra when dealing with variables rather than explicit numbers).
- For example, with numbers we can write the following:

  5 + 9

  (5 + 9) * 3

  -7

  (x + y) / 4

Operators

- In ordinary arithmetic we may use the *operators*

  $+, \quad -, \quad *, \quad /$

  – These work on two numbers and produce a number as result - they are *binary operators.*

- But "minus" also has another sense, as represented by the example

  -7

  – This operator works on one number and produces a number as result - this is a *unary operator*.

- Generally, the result of one operation can be used in an expression involving another operator, as in the example

  *(5 + 9) * 3*

- The operators of the relational algebra also behave like this.

Relational operators

- In 1972, Edward Codd proposed eight operators (though others have been introduced since then).
- All of these can be expressed in terms of five *fundamental* operators:
  - *Restriction*
  - *Projection*
  - *Union (or Addition)*
  - *Difference*
  - *Cartesian Product*
- The three other operators are really shortcuts for frequently used combinations of the fundamental operators. They are:
  - *Join*
  - *(Intersection and Division - these are not covered in this course)*

A note about notation

- There is no standard notation for the relational algebra operators
- The text books vary. Ritchie uses a rather verbose set of keywords, whereas Connolly and Begg stick to Codd's original, very mathematical notation.
- You are allowed to use any recognised syntax in your own assessed work.

Restriction

- The restriction operator works on a single relation *R* and defines a new relation that contains only those rows of *R* that satisfy some specified condition, *C*.
- We denote the restriction operation as follows:

  RESTRICT *R* TO *C*

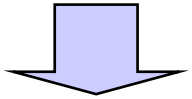
- The restriction operator effectively produces a subset of a relation as shown below. The shading denotes rows that satisfy condition *C*.

Restriction Example

- List all staff with a salary greater than £20,000

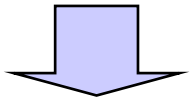
STAFF						
Sno	***Name***	***Position***	***Sex***	***DOB***	***Salary***	***Bno***
SL21	John White	Manager	M	1-Oct-45	30000	B5
SG37	Ann Beech	Snr. Asst.	F	10-Nov-60	12000	B3
SG14	David Ford	Deputy	M	24-Mar-58	18000	B3
SA9	Mary Howe	Assistant	F	19-Feb-70	9000	B7
SG5	Susan Brand	Manager	F	3-Jun-40	24000	B3
SL41	Julie Lee	Assistant	F	13-Jun-65	9000	B5

Required information



RESTRICT STAFF **TO** Salary > 20000

Restriction operation



Sno	***Name***	***Position***	***Sex***	***DOB***	***Salary***	***Bno***
SL21	John White	Manager	M	01-Oct-45	30000	B5
SG5	Susan Brand	Manager	F	03-Jun-40	24000	B3

The result relation

Projection

- The projection operator works on a single relation *R* and defines a relation that contains a subset of the columns of *R* (and eliminates any duplicate rows that may result).
- We denote a projection operation as follows:

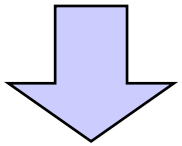
  PROJECT ***ColumnList*** **FROM** ***R***

- The projection operator works as follows. The shaded columns are the ones listed in *ColumnList*.

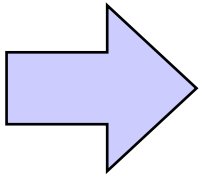
Projection Example 1

- Produce a list of salaries for all staff, showing only the *Sno*, *Name*, and *Salary* details

STAFF						
Sno	***Name***	***Position***	***Sex***	***DOB***	***Salary***	***Bno***
SL21	John White	Manager	M	1-Oct-45	30000	B5
SG37	Ann Beech	Snr. Asst.	F	10-Nov-60	12000	B3
SG14	David Ford	Deputy	M	24-Mar-58	18000	B3
SA9	Mary Howe	Assistant	F	19-Feb-70	9000	B7
SG5	Susan Brand	Manager	F	3-Jun-40	24000	B3
SL41	Julie Lee	Assistant	F	13-Jun-65	9000	B5



PROJECT *Sno, Name, Salary* **FROM** STAFF

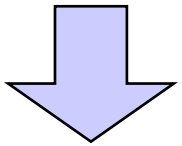


Sno	***Name***	***Salary***
SL21	John White	30000
SG37	Ann Beech	12000
SG14	David Ford	18000
SA9	Mary Howe	9000
SG5	Susan Brand	24000
SL41	Julie Lee	9000

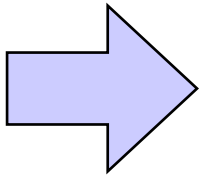
Projection Example 2

- Produce a list of the positions held within the organisation
  - Note that the result has fewer rows than the source relation. Why?

STAFF						
Sno	***Name***	***Position***	***Sex***	***DOB***	***Salary***	***Bno***
SL21	John White	Manager	M	1-Oct-45	30000	B5
SG37	Ann Beech	Snr. Asst.	F	10-Nov-60	12000	B3
SG14	David Ford	Deputy	M	24-Mar-58	18000	B3
SA9	Mary Howe	Assistant	F	19-Feb-70	9000	B7
SG5	Susan Brand	Manager	F	3-Jun-40	24000	B3
SL41	Julie Lee	Assistant	F	13-Jun-65	9000	B5



PROJECT *Position* **FROM** STAFF



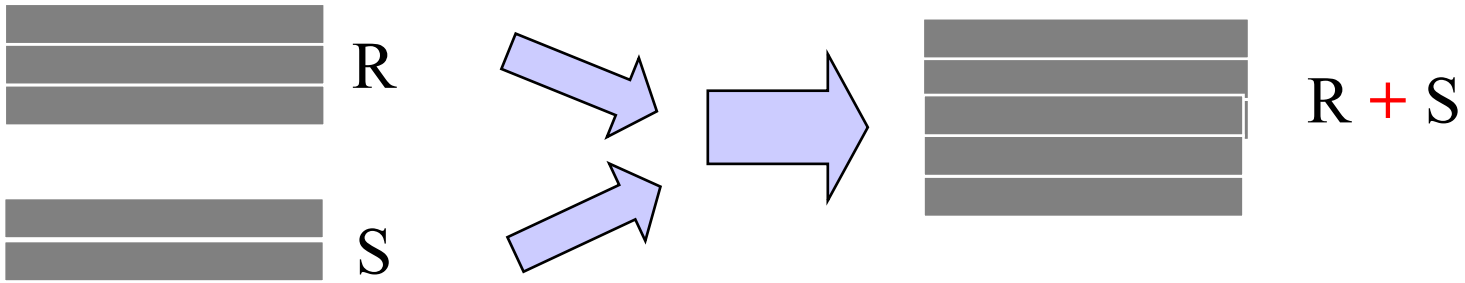
Position
Manager
Snr. Asst.
Deputy
Assistant

Union

- The *union* of two relations *R* and *S* is obtained by pooling their rows into one relation, duplicate rows being eliminated.
  - It operates on two relations.
- The input relations *R* and *S* must be *union-compatible* (explained later).
- We denote a union operation as follows:

$$R + S$$

- The union operation works as follows:

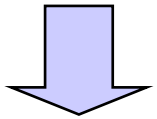


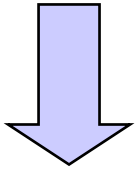
Union Example

- Query: construct a list of all cities where there is **either** a Branch **or** a Property.

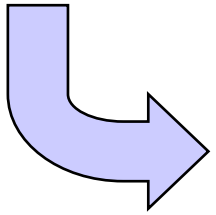
BRANCH				
Bno	**Street**	**Area**	**City**	**Postcode**
B5	22 Deer St	Sidcup	London	SW1 4EH
B7	16 Argyll St	Dyce	Aberdeen	AB2 3SU
B3	163 Main St	Partick	Glasgow	G11 9QX
B4	32 Manse Rd	Leigh	Bristol	BS99 1NZ
B9	56 Clover Dr		London	NW10 6EU

PROPERTY				
Pno	**Street**	**Area**	**City**	**Rent**
PA14	16 Holhead	Dee	Aberdeen	650.00
PL94	6 Argyll St	Kilburn	London	400.00
PG21	18 Dale Rd	Hyndland	Glasgow	600.00





(**PROJECT** City **FROM** BRANCH) + (**PROJECT** City **FROM** PROPERTY)



City
London
Aberdeen
Glasgow
Bristol

Union Compatibility

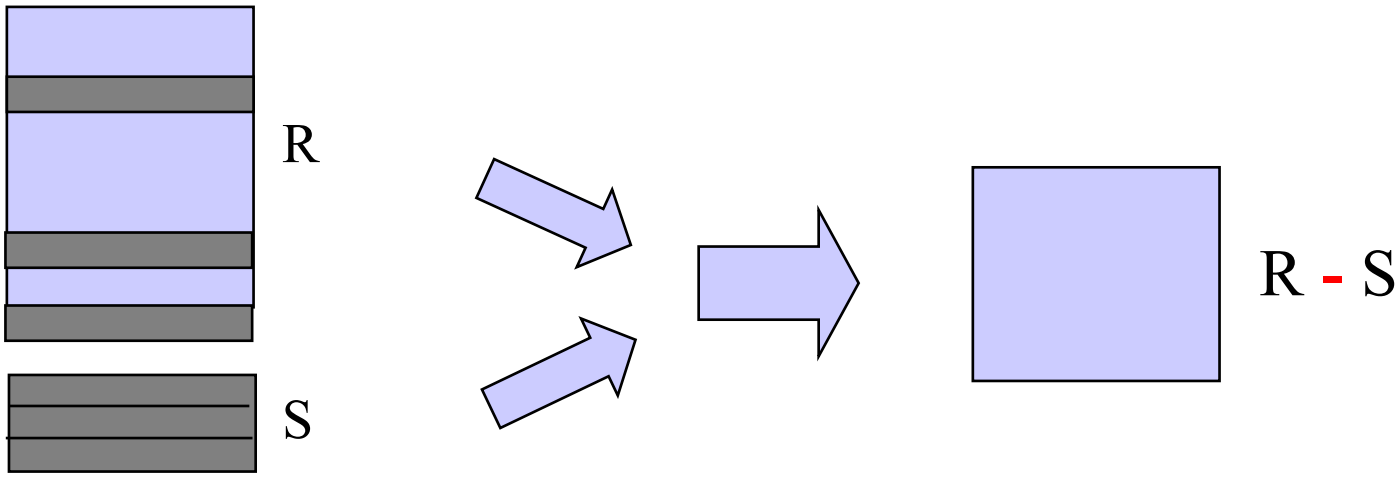
- The union operation can be applied only to relations which are *union compatible*.
  - Both relations must have the same degree (number of attributes).
  - Corresponding attributes must come from the same domain.
- For example, we cannot form the union of *BRANCH* and *PROPERTY*:
  - Both have 5 attributes, so that is ok.
  - But corresponding attributes do not all match (e.g. *Postcode* vs. *Rent*).
- We can sometimes solve union incompatibility problems by using *projection* (as in the previous example).

Difference

- The *difference* operator defines a relation consisting of all rows that are in relation *R*, but not in relation *S*.
  - It operates on two relations.
- As for the union operation, R and S must be *union-compatible*. (Why?)
- We denote a *difference* operation as follows:

  R - S

- The difference operation works as follows:



Difference Example

- Query: construct a list of all cities where there is a Branch **but no** Property.

BRANCH				
Bno	***Street***	***Area***	***City***	***Postcode***
B5	22 Deer St	Sidcup	London	SW1 4EH
B7	16 Argyll St	Dyce	Aberdeen	AB2 3SU
B3	163 Main St	Partick	Glasgow	G11 9QX
B4	32 Manse Rd	Leigh	Bristol	BS99 1NZ
B9	56 Clover Dr		London	NW10 6EU

PROPERTY				
Pno	***Street***	***Area***	***City***	***Rent***
PA14	16 Holhead	Dee	Aberdeen	650.00
PL94	6 Argyll St	Kilburn	London	400.00
PG21	18 Dale Rd	Hyndland	Glasgow	600.00

(**PROJECT** *City* **FROM** BRANCH) **-** (**PROJECT** *City* **FROM** PROPERTY)

City
Bristol

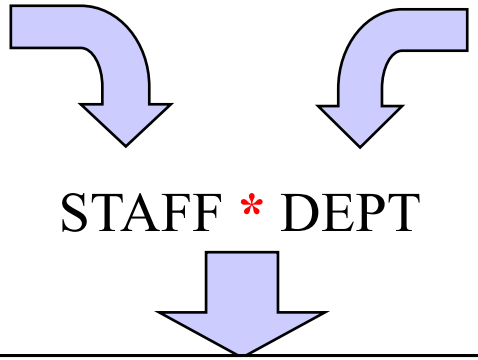
Cartesian Product

- Restriction and Projection allow us to get information out of a *single* relation.
- Union and Difference allow us to manipulate *two* relations vertically (i.e. combine or remove rows).
- We often need to *combine the rows of two relations* in order to relate rows in one relation to the corresponding rows in another relation.
  - This is the purpose of the *Cartesian product* operator
- The Cartesian product operator defines a relation that includes the concatenation of *every* row of relation *R* with *every* row of relation *S*.
  - In other words, it produces every possible combination of the rows of R and S.
- We denote a Cartesian product operation as follows: $R * S$

Cartesian Product Example

- This example does not solve any particular query. It is intended to illustrate how the basic Cartesian product operation works.

STAFF		
Sno	***Name***	***Dno***
SG86	Alan Hamilton	31
SP52	Paul Kingston	49
SJ12	Michael Smith	55



DEPT		
Dno	***Name***	***Rooms***
31	Computing Science	18
49	Management	15
55	Basket-weaving	3

RESULT					
Sno	***STAFF.Name***	***STAFF.Dno***	***DEPT.Dno***	***DEPT.Name***	***Rooms***
SG86	Alan Hamilton	31	31	Computing Science	18
SG86	Alan Hamilton	31	49	Management	15
SG86	Alan Hamilton	31	55	Basket-weaving	3
SP52	Paul Kingston	49	31	Computing Science	18
SP52	Paul Kingston	49	49	Management	15
SP52	Paul Kingston	49	55	Basket-weaving	3
SJ12	Michael Smith	55	31	Computing Science	18
SJ12	Michael Smith	55	49	Management	15
SJ12	Michael Smith	55	55	Basket-weaving	3

Cartesian Product

- In effect, the Cartesian product operation joins the rows from the input relation to form *all possible combinations* of rows.

- How many rows and attributes is that?

- Since the two input relations may have attributes with the same name, it is necessary to avoid ending up with multiple like-named attributes in the result.

- This is avoided by prefixing the names of affected attributes with the names of the input relations:
  - Thus, in the previous example, we end up with *STAFF.Dno* and *DEPT.Dno*.
  - We also have *STAFF.Name* and *DEPT.Name*.

Using Cartesian Product

- We return to the example of the real estate agency.
- Query: list the names of renters who have viewed at least one property, together with the property number in question, and any comment.
- The input relations are shown below:

RENTER		
Rno	***Name***	***Address***
CR76	John Kay	56 High St
CR74	Mike Ritchie	18 Tain St
CR62	Mary Tregear	5 Tarbot Rd

VIEWING				
Pno	***Rno***	***Date***	***Time***	***Comment***
PA14	CR74	21/2/97	09:00	too small
PA14	CR76	21/2/97	11:15	no dining room
PG21	CR74	15/6/97	03:45	
PL94	CR62	18/8/97	09:00	too remote

- To perform this query we require renter names and numbers from the RENTER relation to be combined with the property number and comment information from the VIEWING relation

Using Cartesian Product - I

- Here is a first attempt at a solution:

  R1 = **PROJECT** *Rno, Name* **FROM** RENTER

  R2 = **PROJECT** *Pno, Rno, Comment* **FROM** VIEWING

  RESULT1 = R1 * R2

- This is what the result looks like. What's wrong with it?

RESULT1				
RENTER.Rno	***Name***	***Pno***	***VIEWING.Rno***	***Comment***
CR76	John Kay	PA14	CR74	too small
CR76	John Kay	PA14	CR76	no dining room
CR76	John Kay	PG21	CR74	
CR76	John Kay	PL94	CR62	too remote
CR74	Mike Ritchie	PA14	CR74	too small
CR74	Mike Ritchie	PA14	CR76	no dining room
CR74	Mike Ritchie	PG21	CR74	
CR74	Mike Ritchie	PL94	CR62	too remote
CR62	Mary Tregear	PA14	CR74	too small
CR62	Mary Tregear	PA14	CR76	no dining room
CR62	Mary Tregear	PG21	CR&4	
CR62	Mary Tregear	PL94	CR62	too remote

Using Cartesian Product - II

- The problem is that there are many rows in which *RENTER.Rno* and *VIEWING.Rno* do not match.
- We can solve this problem by using the restriction operator

  RESULT2 = RESTRICT RESULT1 TO *RENTER.Rno = VIEWING.Rno*
- The result of the restriction operation is shown below:

RESULT2				
RENTER.Rno	***Name***	***Pno***	***VIEWING.Rno***	***Comment***
CR76	John Kay	PA14	CR76	no dining room
CR74	Mike Ritchie	PA14	CR74	too small
CR74	Mike Ritchie	PG21	CR74	
CR62	Mary Tregear	PL94	CR62	too remote

- The effect of the restriction is to *eliminate* rows that have been formed by combining unrelated rows in the two original tables.
- Finally, a projection will give us the data we originally sought:

  RESULT = **PROJECT** *Name, Pno, Comment* **FROM** RESULT2

Joins

- The complex sequence of operations on the previous two slides was necessary to produce a *sensible* combination of rows in the result.
- This sequence follows a pattern that is so frequently used that a special operation called *join* has been defined as a shortcut.
- The *join* operator is basically a combination of a Cartesian product and a restriction operation.
- There are a number of variations, some more useful than others.
  - *Natural join - this is the only one we shall cover*
  - *(Equi-join, Outer join, Semi-join - you can read about these in the texts, and you've seen equijoin and outer join in SQL)*

Natural joins

- The natural join operator is a combination of Cartesian Product, Restriction, and Projection.
- We denote a natural join on two relations R and S as follows:

$$R \bowtie S$$

- In effect, the natural join operator does the following:
  - It first forms the Cartesian Product of R and S
  - It then Restricts the result to one in which common attributes from R and S have the *same* value. (Usually, the common attributes are in fact primary and foreign keys.)
  - Finally, it applies the Projection operator so that each of the common attributes from R and S appears only *once* in the final result.

Natural Join Example

- Repeating the previous example using a natural join gives the following result.

RENTER ⋈ VIEWING

Rno	*Name*	*Address*	*Pno*	*Date*	*Time*	*Comment*
CR76	John Kay	56 High St	PA14	21/02/97	11:15	no dining room
CR74	Mike Ritchie	18 Tain St	PA14	21/02/97	09:00	too small
CR74	Mike Ritchie	18 Tain St	PG21	15/06/97	03:45	
CR62	Mary Tregear	5 Tarbot Rd	PL94	18/08/97	09:00	too remote

- The common attribute is the *Rno* field.
  - It is the primary key of the *RENTER* relation.
  - It is a foreign key in the *VIEWING* relation
- The *Rno* field occurs only once in the result.

Solving Real Queries

- As it happens, many real queries can be solved using a standard formula of *restrict*, *project*, and *natural join*.
- Here is another example:
- Query: List the names of the members of staff who work in the Computing Science department.

DEPT		
Dno	***Name***	***Rooms***
31	Computing Science	18
49	Management	15
55	Basket-weaving	3

STAFF		
Sno	***Name***	***Dno***
SG86	Savi Maharaj	31
SP52	Paul Kingston	49
ST22	Richard Bland	31

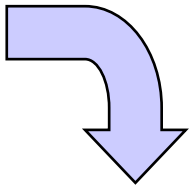
- To solve this query we use only these three operators:

Restrict, Project, Natural Join

Solution to Query - I

- First we use the restrict operator to obtain only the information about the Computing Science department:

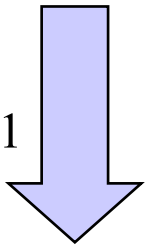
R1 = **RESTRICT** DEPT **TO** *Name* = 'Computing Science'



R1		
Dno	***Name***	***Rooms***
31	Computing Science	18

- Using this result, we can now project out only the *Dno* column for natural join.

R2 = **PROJECT** *Dno* **FROM** R1



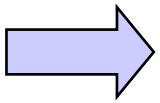
R2
Dno
31

- We will then use this result to perform a natural join with the STAFF table….

Solution to Query - II

- The natural join is as follows:

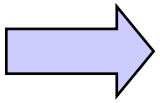
$$R3 = R2 \bowtie STAFF$$



R3		
Sno	***Name***	***Dno***
SG86	Savi Maharaj	31
ST22	Richard Bland	31

- The query asked only for the names of staff, so to finish this query, we need to project this information from R3.

RESULT = **PROJECT** *Name* **FROM** R3



RESULT
Name
Savi Maharaj
Richard Bland

- The solution took four steps.
  - Many queries can be answered in this way.
  - The first *project* operation wasn't strictly necessary, so we could have done it in three steps.

Other Operators

- There are other operators that we have not covered.
- For example:
  - *Intersection*
  - *Divide*
  - Other types of *join*
- You may find these in textbooks.
- We do not cover them as they are not often used.
  - Furthermore, we can do everything with the 5 basic operators (restriction, projection, join, union, difference) so these other *advanced* operators are really only *shortcuts*.

Using the Relational Algebra

- We have covered the majority of operations commonly used in the relational algebra.
- As our examples have shown, we can combine operations to produce answers to queries.
- Most queries can be solved by a combination of *restriction*, *projection*, and *join*.
- However, some complex queries may also require *union* or *difference*.
- Consider the data and the queries on the following slides:

Sample Relations

RENTER				
Rno	***Fname***	***Lname***	***Address***	***Phone***
CR76	John	Kay	56 High St	0171-774-5632
CR56	Aline	Stewart	64 Fern Dr	0141-848-1825
CR74	Mike	Ritchie	18 Tain St	01475-392178
CR62	Mary	Tregear	5 Tarbot Rd	*01224-196720*

VIEWING			
Rno	***Pno***	***Date***	***Comment***
CR56	*PA14*	*20-Apr-95*	*too small*
CR76	*PG4*	*20-Apr-95*	*too remote*
CR56	*PG4*	*26-May-95*	
CR62	*PA14*	*14-May-95*	*no dining room*
CR56	*PG36*	*28-Apr-95*	

PROPERTY							
Pno	***Street***	***Area***	***City***	***Postcode***	***Type***	***Rooms***	***Rent***
PA14	16 Holhead	Dee	Aberdeen	AB7 5SU	House	6	650.00
PL94	6 Argyll St	Kilburn	London	NW2	Flat	4	400.00
PG4	6 Lawrence St	Partick	Glasgow	G11 9QX	Flat	3	350.00
PG36	2 Manor Rd		Glasgow	G32 4QX	Flat	3	375.00
PG21	18 Dale Rd	Hyndland	Glasgow	G12	House	5	600.00

A Complex Query

- List the names of renters who have viewed *all properties with three rooms*.
- (Hint: Consider the different steps required, then combine them.)

Steps

1 List the names and numbers of all renters.

2 List the numbers of properties with 3 rooms.

3 List all possible combinations of renters and 3-room properties.

4 List the names and numbers of renters who have viewed 3-room properties (including the property numbers).

5 Use the results of steps 3 and 4 to list the renter and 3-room property combinations that have *not* actually happened.

6 Using the result of step 5, list the names and numbers of renters who have not viewed all 3-room properties.

7 Finally, use the results of steps 1 and 6 to solve the query!

End of Lecture

Would you like to ask anything?

Don’t forget to read the notes again, and Ritchie Chapter 2.