Relational Algebra

Databases, Aarhus University

Ira Assent

Intended learning outcomes

- Be able to
  - Read and write relational algebra expressions
  - Map relational algebra expressions to SQL counterparts

ira@cs.au.dk

Recap: Recovery

- In case of crash, different recovery actions necessary depending on policies during operation
  - Undo/Redo: redo transactions where log has "start" and "commit", undo transactions where log has "start" but no "commit"
    - Maximize efficiency during normal operation, more recovery work, requires before/after information in log
  - No-Undo/Redo: no output on disk until commit log on stable storage
    - Database outputs must wait, more work at commit time, faster during recovery: no undo, no before information in log
  - Undo/No-Redo: changes to disk before commit, requires that write entry first be output to (stable) log
    - No after images are needed in log, no redo, many I/O for committed write
  - No-Undo/No-Redo: changes only on shadow pages (copies), on commit changes written to database in a single atomic action
    - Recovery instantaneous, nothing to be done, but access to stable storage indirect, original layout of data destroyed, concurrent transactions difficult to support

ira@cs.au.dk

ACID: What is recovery mostly concerned with?

A. A, I
B. A, C
C. A, D
D. I, C
E. I, D
F. C, D

ira@cs.au.dk

Foundations of SQL

- So far, we have expressed our queries in SQL
  - Declarative language to describe properties of the result
  - SQL is de facto standard in relational DBMS
    - Convenient language for human users
    - Historically, developed after first relational DBMS
      - In 1970: Edgar Codd describes relational algebra for DBMS
      - IBM implemented SEQUEL (Structured English QUEry Language), later SQL

- We will look at relational algebra
  - to understand the foundations of SQL
- Then turn to relational calculus
- From basis for query execution and query optimization

ira@cs.au.dk

Relational Algebra

- Relational algebra is basic set of operations for the relational model
  - Formal foundation for relational model operations
  - Relational algebra expression (sequence of operations) represents the resulting relation of the database query
- A sequence of relational algebra operations forms relational algebra expression
  - As in SQL: use basic building blocks to construct your query or your statement
- A relation is like a table where
  - all columns have the same generic type
  - no other constraints are imposed
  - We implicitly allow permutations of the attributes – no order imposed!
  - no duplicates are allowed: set model!
- A database relation on a data set D consists of
  - a schema of attribute names ($a_1, a_2, \ldots, a_n$)
  - a finite n-ary relation on D, a subset of D^n
    - $R = \{(r_{i1}, r_{i2}, \ldots, r_{in}) \mid i=1\ldots m\}$ where n is the number of attributes, m is the number of rows

a_1	a_2	…	a_n
r_{11}	r_{12}	...	r_{1n}
r_{1m}	r_{2m}	...	r_{mn}

ira@cs.au.dk

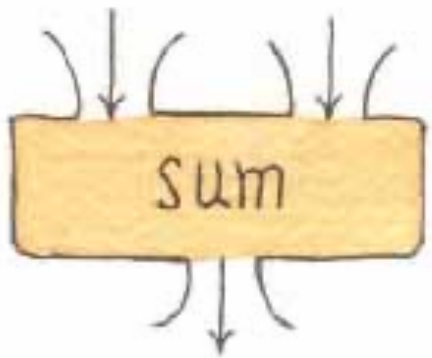
Mathematical relations

- An n-ary relation on a set S is a subset of S^n
- Examples
  - $\leq$ is a binary relation on $\mathbf{R}$, a subset of $\mathbf{R}\times\mathbf{R}$

    { (1.2, 3.4), (34, 117.363), (–53, 0.1234), ... }
  - *divorced from* is a binary relation on people

    { (Miley, Liam), (Liam, Miley), (Shakira, Gerard), ... }
  - *sum* is a ternary relation on $\mathbf{N}$, a subset of $\mathbf{N}\times\mathbf{N}\times\mathbf{N}$

    { (3,5,8), (23,14,37), (0,123,123), (42,87,129), ... }



ira@cs.au.dk

What is an Algebra?

- An algebra consists of values, operators and rules
  - Examples
    - integers with +, –, ×
    - sets with ∪, ∩, \, ×
    - Database relations with query operators

      union ∪ , intersection ∩ , difference \ , projection π , renaming ρ , selection σ , Cartesian product × , natural join ⋈

      - Closure: relation in – relation out, can be nested
        - Again, exactly as for SQL, where the output of a SQL query can be used in another SQL query (nested)
      - Specify retrieval requests as relational algebra expressions
        - Result is a relation that represents result of database query
        - provide an abstract model of database queries

ira@cs.au.dk

Selection

- Specify a subset of tuples that satisfy a condition
- Notation: $\sigma_C(R)$
  - C is a condition on the attributes of R
    - Composed of attribute names, comparison operators and values or other attribute names
  - The resulting schema is unchanged
  - The relation part is: $\{ r \mid r \in R \wedge C(r) \}$
- Example: $\sigma_{type=\text{'projector'}}$(Rooms) has unchanged schema (room, type) and returns only tuples that meet condition

corresponds to condition in `WHERE` clause

Rooms

room	type
StoreAud	projector
StoreAud	whiteboard
Ho-017	mini-fridge

room	type
StoreAud	projector

ira@cs.au.dk

Selection

- $\sigma_C(R)$
  - C is a condition of the attributes of R
  - The resulting schema is unchanged
  - The relation part is: $\{ r \mid r \in R \land C(r) \}$

Participants

meetid	pid	status
34716	StoreAud	a
34716	ira	a
42835	zoffe	d

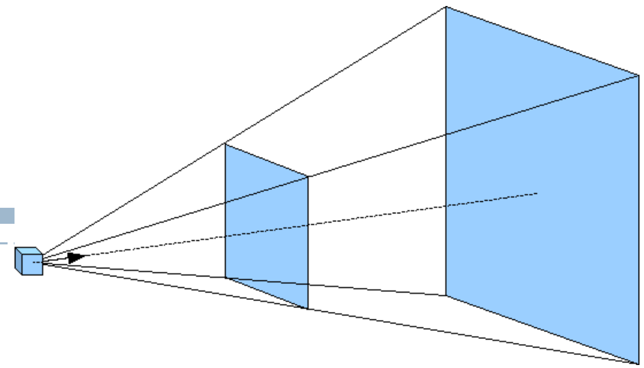
In RA: find information on participants who have accepted meeting 34716

meetid	pid	status
34716	StoreAud	a
34716	ira	a

How do you express this in SQL?

ira@cs.au.dk

Projection



- Specify a subset of attributes
- Notation: $\pi_{a_1,\ldots,a_n}(R)$

corresponds to attributes listed in `SELECT` clause

  Where schema of R is $(a_1,\ldots,a_n,b_1,\ldots,b_m)$

  The schema of the result is $(a_1,\ldots,a_n)$

  The result relation is $\{ (d_1,\ldots, d_n) \mid (d_1, \ldots, d_{n+m}) \in R \}$

- Note: as we are working with sets, duplicates are eliminated, result are all distinct tuples
- Example: $\pi_{group,office}$(People) has schema (group,office) and each tuple only has values for these two attributes

In RA: find all names of people

People

userid	name	group	office
ira	Ira Assent	vip	Ny-357
aas	Annika Schmidt	phd	NULL
jan	Jan Christensen	tap	Ho-017

group	office
vip	Ny-357
phd	NULL
tap	Ho-017

ira@cs.au.dk

Projection

Find information on name of the event and meeting id where aas is the owner

A. $\pi_{meetid,topic,owner='aas'}$ (Meetings)
B. $\pi_{meetid,topic}(\sigma_{owner='aas'}$(Meetings))
C. $\sigma_{owner='aas'}(\pi_{meetid,topic}$ (Meetings))
D. $\pi_{meetid,topic}$(Meetings)

- Projection: $\pi_{a_1,\ldots,a_n}(R)$

  Assume the schema of R is $(a_1,\ldots,a_n,b_1,\ldots,b_m)$

  The schema of the result is $(a_1,\ldots,a_n)$

  The result relation is $\{ (d_1,\ldots, d_n) \mid (d_1, \ldots, d_{n+m}) \in R \}$

Meetings

meetid	date	owner	topic
34716	2021-08-28	ira	dDB
34717	2021-08-22	ira	dDB
42835	2022-04-18	aas	TA meeting

meetid	topic
42835	TA meeting

How do you express this in SQL?

ira@cs.au.dk

Sequences of Operations

- To nest operations, use sequences
- Two ways of expressing nesting
  - In-line expression (preferred)

$$\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno}=5}(\text{EMPLOYEE}))$$

  - Sequence of operations
    - Use assignment operation to explicitly name intermediate relations
      - Sometimes used in textbook examples

$$\text{DEP5\_EMPS} \leftarrow \sigma_{\text{Dno}=5}(\text{EMPLOYEE})$$
$$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Salary}}(\text{DEP5\_EMPS})$$

Renaming

corresponds to AS

- $\rho_{S(b1,b2,\ldots,bn)}(R)$ to rename relation and attributes, $\rho_S(R)$ to rename relation, $\rho_{(b1,b2,\ldots,bn)}(R)$ to rename attributes
  - S new relation name, b1,b2,…bn new attribute names
- Or use $\rho_{a \to b}(R)$ to rename some of the attributes
  - The name a must occur as a_i in the schema of R
  - The name b must not occur in the schema of R
- Schema of the result: $(a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_n)$
  - The result relation is unchanged
- Example: $\rho_{\text{lokale, antal}}(\text{Rooms}) = \rho_{\text{room}\to\text{lokale, capacity}\to\text{antal}}(\text{Rooms})$ has schema Rooms(lokale, antal), data in table unchanged

room	capacity
Ny-357	6
StoreAud	286

lokale	antal
Ny-357	6
StoreAud	286

From Meetings, find the "DB people" as everyone who has the topic "dDB"

meetid	date	owner	topic
34716	2021-08-28	ira	dDB
42835	2022-04-18	aas	TA meeting

Cartesian Product

Dish

dish	price
GiantBurger	40
TofuDelight	35

Drinks

drink	size
Beer	0.5
GreenTea	0.3

- Cartesian Product: R × S
  - Also called cross product or cross join
- The new schema is $(a_1, \ldots, a_m, b_1, \ldots, b_n)$
  - if R has schema $(a_1, \ldots, a_m)$ and S has schema $(b_1, \ldots, b_n)$
- The relation part is

$$\{ (c_1, \ldots, c_{m+n}) \mid (c_1, \ldots, c_m) \in R \wedge (c_{m+1}, \ldots, c_{m+n}) \in S \}$$

corresponds to listing more than one table in `FROM` clause

dish	price	drink	size
GiantBurger	40	Beer	0.5
TofuDelight	35	GreenTea	0.3
GiantBurger	40	GreenTea	0.3
TofuDelight	35	Beer	0.5

- Example: for Dish(dish,price) and Drinks(drink, size), Dish x Drinks has schema (dish, price, drink, size) and all possible combinations form the relation

Find people names and the capacity of the room they are in

People

userid	name	group	office
ira	Ira Assent	vip	Ny-357

Room

room	capacity
Ny-357	6

Union, Intersection, Difference

- The arguments must have the same schema
- The result has again that schema
  - $R \cup S$
  - $R \cap S$
  - $R \setminus S$

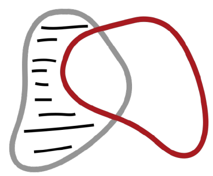
R

a_1	a_2	…	a_n

$\cup$ $\cap$ $\setminus$

a_1	a_2	…	a_n

S



Student

id	name
1	Mads
2	Ann

Teacher

id	name
3	Rie
4	Kurt

- They compute the set operations on the relations

correspond to set operations in SQL

- Example: Student $\cup$ Teacher works for tables Student and Teacher with identical schema (id, name); result schema is again (id, name); result tuples are the union of all tuples in the two tables
  - Again, duplicate tuples eliminated

id	name
1	Mads
2	Ann
3	Rie
4	Kurt

We'd like to see the names of all rooms with a projector that are not broken, using relations Rooms(room, equipment) and BrokenRooms(room, equipment)

A Complete Set of Relational Algebra Operations

- Set of relational algebra operations {σ, π, ∪, ρ, –, ×} is a complete set
  - Any relational algebra operation can be expressed as a sequence of operations from this set

  - We introduce additional operators for convenience
    - In particular, joins and division

    - In SQL, we can similarly express joins as cartesian products with `WHERE` clause conditions on join attributes or by using SQL syntax like "`JOIN ON`" etc
      - In relational algebra, cartesian product with selection

ira@cs.au.dk

JOIN

- Combine related tuples from two relations into single "longer" tuples
  - As in SQL
  - Denoted by $\bowtie$
  - General join condition of the form <condition> AND <condition> AND...AND <condition>
  - Example:

$$\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE}$$
$$\text{RESULT} \leftarrow \pi_{\text{Dname, Lname, Fname}}(\text{DEPT\_MGR})$$

Find people names and the capacity of the room they are in

People

userid	name	group	office
ira	Ira Assent	vip	Ny-357

Room

room	capacity
Ny-357	6

MeetStats

Meetid	pcount
34716	83
42835	5

Room

room	capacity
Ny-357	6
Ada-333	26
StoreAud	286

Formalizing joins

- Theta join $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  - A general join with a condition on the involved relations
  - Each <condition> of the form $A_i\ \theta\ B_j$
  - A_i is an attribute of R, B_j is an attribute of S, A_i and B_j have the same domain, θ (theta) is one of the comparison operators $=, <, \leq, >, \geq, \neq$
  - Corresponds to $\sigma_{\theta}(R_1 \times S)$: theta condition on cartesian product
  - Room $\bowtie_{capacity>pcount}$ MeetStats

```
SELECT DISTINCT X1, …, Xk
  FROM R1, …, Rn
  WHERE θ
```

meetid	pcount	room	capacity
34716	83	StoreAud	286
42835	5	Ny-357	6
42835	5	Ada-333	26
42835	5	StoreAud	286

Find the userids of meeting owners who are phd students.

People

userid	name	group	office
ira	Ira Assent	vip	Ny-357

Meetings

meetid	date	owner	topic

Variations of JOIN

- EQUIJOIN
  - Only = comparison operator used
  - Pairs of attributes that have identical values in every tuple
  - E.g. matching MGR_SSN to SSN
- NATURAL JOIN
  - Denoted by *
  - Equijoin on attributes of same name
  - But removes second (superfluous) attribute in an EQUIJOIN condition
  - E.g. matching attributes NAME in tables Student and TA
- Same as in SQL

Natural Join

Prices

dish	price
GiantBurger	40
TofuDelight	35

$\bowtie$

Restaurants

dish	restaurant
GiantBurger	BigJoe
TofuDelight	LittleIndia

dish	price	restaurant
GiantBurger	40	BigJoe
TofuDelight	35	LittleIndia

- R * S

  new schema is $(a_1, ..., a_k, c_1, ..., c_n, b_1, ..., b_m)$

  if R has schema $(a_1, ..., a_k, c_1, ..., c_n)$

  and S has schema $(c_1, ..., c_n, b_1, ..., b_m)$

  and $\{a_i\} \cap \{b_i\} = \varnothing$

- The relation part is

corresponds to `NATURAL JOIN`

$$\{ (d_1, ..., d_k, e_1, ..., e_n, f_1, ..., f_m) \mid (d_1, ..., d_k, e_1, ..., e_n) \in R \wedge (e_1, ..., e_n, f_1, ..., f_m) \in S \}$$

Example: Prices * Restaurants for Prices(dish, price) and Restaurants (dish, restaurant) yields result schema (dish, price, restaurant) and "matching" tuples combined (same join attribute name(s) and value(s))

We'd like to find "Møder" as the ids of meetings and those of their participants (if they accepted)

Meetings

meetid	date	owner	topic
34716	2021-08-28	ira	DB

Participants

meetid	pid	status
34716	StoreAud	a

ira@cs.au.dk

Operating operators

How does R * S behave?

A. R * S = R ∩ S when the schemas share more than one attribute and R * S = R × S when the schemas share a joint attribute

B. R * S = R ∩ S when the schemas share a joint attribute and R * S = R × S when the schemas share more than one attribute

C. R * S = R ∩ S when the schemas are identical and R * S = R × S when the schemas are disjoint

D. R * S = R ∩ S when the schemas are disjoint and R * S = R × S when the schemas are identical

R * S schema $(a_1, \ldots, a_k, c_1, \ldots, c_n, b_1, \ldots, b_m)$

R schema $(a_1, \ldots, a_k, c_1, \ldots, c_n)$, S schema $(c_1, \ldots, c_n, b_1, \ldots, b_m)$, $\{a_i\} \cap \{b_j\} = \emptyset$

R * S relation $\{ (d_1, \ldots, d_k, e_1, \ldots, e_n, f_1, \ldots, f_m) \mid (d_1, \ldots, d_k, e_1, \ldots, e_n) \in R \wedge (e_1, \ldots, e_n, f_1, \ldots, f_m) \in S \}$

ira@cs.au.dk

Another query

In which meetings (meetid) do the owners participate (accepted)?

Bonus: how do you write that in SQL?

Meetings

meetid	date	owner	topic
34716	2021-08-28	ira	dDB
34717	2022-03-22	ira	dDB
42835	2022-04-18	aas	TA meeting

Participants

meetid	pid	status
34716	StoreAud	a
34716	ira	a
42835	zoffe	d

ira@cs.au.dk

The DIVISION Operation

- Convenient abbreviation for some queries with "all" quantification
  - Find those tuples in first relation that contain all tuples in second relation
  - Attributes in second relation are a proper subset of attributes in first relation
  - Example: retrieve names of employees who work on all the projects that 'John Smith' works on

a) SSNS is SSN_PNOS divided by SMITH_PNOS

Find all tuples in R where all tuples in S occur

b) T is R divided by S

(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

Division formally

Students

name	course
Mads	DB
Mads	Prog
Ann	DB
Rie	DB
Rie	Prog
Kurt	Prog

Mandatory

course
DB
Prog

name
Mads
Rie

- $A \div B = \pi_x(A) - \pi_x((\pi_x(A) \times B) - A)$
  - attributes of B are a proper subset of attributes of A
  - X is set of attributes in A, but not in B
  - returns tuples from A which match all B's tuples in all attributes
  - Result schema is (attributes of A – attributes of B) = X
- Example: Students who finished all mandatory classes
  - Students÷Mandatory for Students(name, course) and Mandatory(course) yields result schema (name) and returns all tuples that match all entries in Mandatory
- Try out the "translation" using projection, set difference

We'd like a list of dishes that contain all my favorite ingredients using relations Menu(dish, ingredient) and Favorites(ingredient)

Menu

dish	ingredient
GiantBurger	peanut
TofuDelight	peanut
TofuDelight	soy

Favorites

ingredient
peanut
soy

dish
TofuDelight

ira@cs.au.dk

Remember this slide? (Funny) terminology

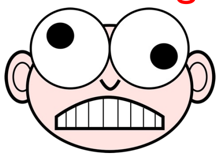
- The basic form of an SQL query

`SELECT` *desired attributes*
   `FROM` *one or more tables*
   `WHERE` *condition about the involved rows*

Projection: projecting to attributes

Cartesian Product / Join
(in combination with where clause):
combining tables

Selection condition: selecting rows



Should make a bit more sense now:

`SELECT` - π projection

`FROM` - $\times$ cartesian product

`WHERE` - σ selection

ira@cs.au.dk

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{(\text{<join attributes 1>}),\ (\text{<join attributes 2>})} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{(\text{<join attributes 1>}),\ (\text{<join attributes 2>})} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Additional Relational Operations

Students

name	course
Mads	DB
Mads	Prog
Ann	DB
Rie	DB
Rie	Prog
Kurt	Prog

- Generalized projection $\pi_{F1,\, F2,\, \ldots,\, Fn}\,(R)$
  - Allows functions of attributes to be included in the projectio
    - Aggregate functions and grouping
      - E.g. SUM, AVERAGE, MAXIMUM, and MINIMUM
- Group tuples by the value of some of their attributes
  - Apply aggregate function independently to each group

$$\text{<grouping attributes>}\ \Im\ \text{<function list>}\ (R)$$

a. $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}(_{\text{Dno}}\ \Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE}))$.

b. $_{\text{Dno}}\ \Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

c. $\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

(a)

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

(b)

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

(c)

Count_ssn	Average_salary
8	35125

ira@cs.au.dk

Recursive Queries

- Operation applied to **recursive relationship** between tuples of same type
  - Join table with itself
  - Example: find ssns of all employees directly supervised by James Borg

$$BORG\_SSN \leftarrow \pi_{Ssn}(\sigma_{Fname='James' \text{ AND } Lname='Borg'}(EMPLOYEE))$$
$$SUPERVISION(Ssn1, Ssn2) \leftarrow \pi_{Ssn,Super\_ssn}(EMPLOYEE)$$
$$RESULT1(Ssn) \leftarrow \pi_{Ssn1}(SUPERVISION \bowtie_{Ssn2=Ssn} BORG\_SSN)$$

  - Find Ssns of all employees supervised by someone supervised by James Borg

$$RESULT2(Ssn) \leftarrow \pi_{Ssn1} (SUPERVISION \bowtie_{Ssn2=Ssn} RESULT1)$$

  - Cannot specify a general query to find Ssns of anyone supervised by James Borg at any level – directly, one level below, two levels below,…

SUPERVISION

(Borg's Ssn is 888665555)

Ssn1 (Ssn)	Ssn2 (Super_ssn)
123456789	333445555
333445555	888665555
999887777	987654321
987654321	888665555
666884444	333445555
453453453	333445555
987987987	987654321
888665555	null

RESULT1

Ssn
333445555
987654321

(Supervised by Borg)

RESULT2

Ssn
123456789
999887777
666884444
453453453
987987987

(Supervised by Borg's subordinates)

Recursive Closure

- Cannot compute the transitive closure of a binary relation R

$$R^{\infty} = \{ (x_1,x_k) \mid \exists x_1,\ldots,x_{k-1} \ ((x_i,x_{i+1}) \in R) \}$$

  - Means that Relational Algebra is not Turing complete
  - But that also means easier to optimize queries
- Another example:
  - Which cities can be reached from Copenhagen in one or more flights?
  - Cannot be specified for arbitrarily many hops
- Arbitrary transitive closure operation has been proposed to compute recursive relationship (also for SQL in SQL3 standard)

from	to
Cph	Madrid
Rome	London
Madrid	Athens
Athens	Rome
…	…

ira@cs.au.dk

OUTER JOIN Operations

- **Outer joins** ⟗
  - Keep all tuples in *R*, or all those in *S*, or all those in both relations regardless of whether or not they have matching tuples in the other relation
    - Similar to SQL
  - **Types**
    - **LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN**
  - Example:

$$\text{TEMP} \leftarrow (\text{EMPLOYEE} ⟕_{\text{Ssn=Mgr\_ssn}} \text{DEPARTMENT})$$

$$\text{RESULT} \leftarrow \pi_{\text{Fname, Minit, Lname, Dname}}(\text{TEMP})$$

ira@cs.au.dk

What is equivalent to the following SQL query?

SELECT DISTINCT Students.name
  FROM Students, TA
  WHERE Students.name=TA.name;

A. Students $\bowtie_{\text{Students.name=TA.name}}$ TA
B. $\sigma_{\text{Students.name=TA.name}}$(Students $\times$ TA)
C. $\pi_{\text{Students.name}}$ (Students $*$ TA)
D. π_{name}(Students) $\cap$ π_{name} (TA)

Students

id	name
1	Mads
2	Ann
3	Rie
4	Kurt

TAs

id	name	course	TA class
3	Rie	DB	DA0
4	Kurt	DB	DA99
5	Ann	Prog	DA42
6	Tore	Alg	DA010

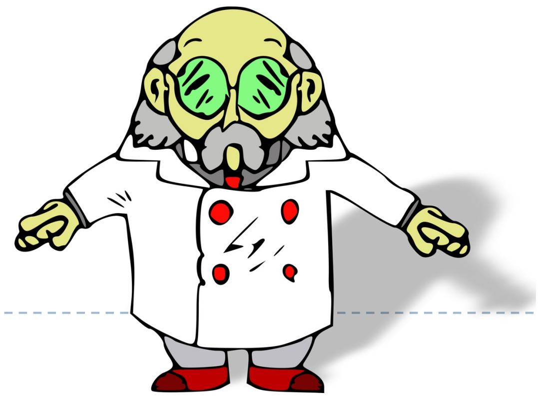
Examples of Queries in Relational Algebra

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

$$\text{RESEARCH\_DEPT} \leftarrow \sigma_{\text{Dname='Research'}}(\text{DEPARTMENT})$$
$$\text{RESEARCH\_EMPS} \leftarrow (\text{RESEARCH\_DEPT} \bowtie_{\text{Dnumber=Dno}} \text{EMPLOYEE})$$
$$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Address}}(\text{RESEARCH\_EMPS})$$

As a single in-line expression, this query becomes:

$$\pi_{\text{Fname, Lname, Address}} (\sigma_{\text{Dname='Research'}}(\text{DEPARTMENT} \bowtie_{\text{Dnumber=Dno}}(\text{EMPLOYEE}))$$



ira@cs.au.dk

Examples of Queries in Relational Algebra (cont'd.)

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

$\text{STAFFORD\_PROJS} \leftarrow \sigma_{\text{Plocation='Stafford'}}(\text{PROJECT})$
$\text{CONTR\_DEPTS} \leftarrow (\text{STAFFORD\_PROJS} \bowtie_{\text{Dnum=Dnumber}} \text{DEPARTMENT})$
$\text{PROJ\_DEPT\_MGRS} \leftarrow (\text{CONTR\_DEPTS} \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE})$
$\text{RESULT} \leftarrow \pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}}(\text{PROJ\_DEPT\_MGRS})$

Query 3. Find the names of employees who work on *all* the projects controlled by department number 5.

$\text{DEPT5\_PROJS} \leftarrow \rho_{(\text{Pno})}(\pi_{\text{Pnumber}}(\sigma_{\text{Dnum=5}}(\text{PROJECT})))$
$\text{EMP\_PROJ} \leftarrow \rho_{(\text{Ssn, Pno})}(\pi_{\text{Essn, Pno}}(\text{WORKS\_ON}))$
$\text{RESULT\_EMP\_SSNS} \leftarrow \text{EMP\_PROJ} \div \text{DEPT5\_PROJS}$
$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{RESULT\_EMP\_SSNS} * \text{EMPLOYEE})$

ira@cs.au.dk

Examples of Queries in Relational Algebra (cont'd.)

Query 6. Retrieve the names of employees who have no dependents.

This is an example of the type of query that uses the MINUS (SET DIFFERENCE) operation.

$\text{ALL\_EMPS} \leftarrow \pi_{\text{Ssn}}(\text{EMPLOYEE})$
$\text{EMPS\_WITH\_DEPS(Ssn)} \leftarrow \pi_{\text{Essn}}(\text{DEPENDENT})$
$\text{EMPS\_WITHOUT\_DEPS} \leftarrow (\text{ALL\_EMPS} - \text{EMPS\_WITH\_DEPS})$
$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{EMPS\_WITHOUT\_DEPS} * \text{EMPLOYEE})$

Query 7. List the names of managers who have at least one dependent.

$\text{MGRS(Ssn)} \leftarrow \pi_{\text{Mgr\_ssn}}(\text{DEPARTMENT})$
$\text{EMPS\_WITH\_DEPS(Ssn)} \leftarrow \pi_{\text{Essn}}(\text{DEPENDENT})$
$\text{MGRS\_WITH\_DEPS} \leftarrow (\text{MGRS} \cap \text{EMPS\_WITH\_DEPS})$
$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{MGRS\_WITH\_DEPS} * \text{EMPLOYEE})$

ira@cs.au.dk

Summary

- Intended learning outcomes
  - Be able to
    - Read and write relational algebra expressions
    - Map relational algebra expressions to SQL counterparts

ira@cs.au.dk

What was this all about?

Guidelines for your own review of today's session

- Relational algebra is used to…
  - Its basic operators are related to SQL clauses as follows…
  - Sequences of operators are created in one of two ways…
- The following operators are complete…
  - Which means…
  - Still, additionally we have further operators…
- Relational algebra and SQL do have some limitations…
- The set model in relational algebra differs from the SQL data model in that…

ira@cs.au.dk