5 The Relational Data Model: Algebraic operations on tabular data

5.1 Foundation of relational languages

5.2 Relational Algebra operations

5.3 Relational Algebra: Syntax and Semantics

5.4. More Operators

5.5 Special Topics of RA

6 The Relational Data Model: Logic foundation of Data Manipulation

Not presented in class!

Kemper / Eickler: 3.4, 4.6+7; Elmasri /Navathe: chap. 74-7.6,
Garcia-Molina, Ullman, Widom: chap. 5, D. Maier Theory of RDB (Online Book -> Lit.)

Context

Freie Universität Berlin

Part 1: Designing and using database

Database Design:
- developing a relational database schema

Design:
- formal theory

Data handling in rela-:
tional databases
-Algebra, -Calculus, SQL/DML

Using the Database
from application progs

Physical Schema

Part 2: DWH, IR .. Transactions

5.1 Foundation of relational languages

Data Model:

Language for **definition** *and* **handling** (manipulation) **of data**

Languages for data handling:

- **Relational Algebra** (RA) as a semantically well defined **applicative language**
- **Relational tuple calculus** (domain calculus): **predicate logic** interpretation of data and queries
- **SQL / DML** ('Sequel') – based on **RA and calculus**

SQL: very important in practice

Relational Languages

Goal of DB language design:

Simple and powerful expressions for querying a database

Language should be <u>declarative</u> ("descriptive")

Historically: “Make query formulation ‘as easy as in natural language’ “

More serious: Queries should be independent of representation of data and implementation aspects (Codd's principle).

Relational Algebra

Algebra objects:

Relations (tables)
$R_1(a_1,...,a_n)$, $R_n(b_1,...,b_m)$ over domains ai, bj,..

Algebra operators:

Operators : transform one or more relations into a relation:
$\tau : R_{i1}(...) \times ... \times R_{ik}(...) \rightarrow R_{im}(...)$

Relational Algebra: only unary and binary operators

Relational Algebra

Freie Universität Berlin

```
City (name r_id  popul .. )
      Oslo    ..  1.31
      Berlin  B   3.47...
      Vienna  ..   957
```

```
Country (c_id r.id capital..)
         'GER' 'B'  Berlin
          'AU' 'V'     Vienna
          ...
```

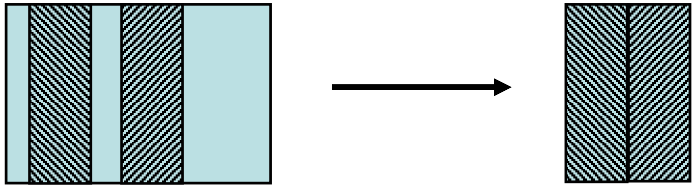
"Name and population of capital of 'Germany' "

$\bowtie$

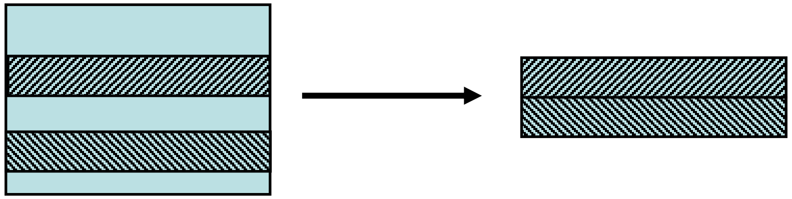
```
NAME                                   POPULATION
-------------------------------------- ------------------------
Berlin                                 3472009
```

Basic Operations informally (from chapter 4)

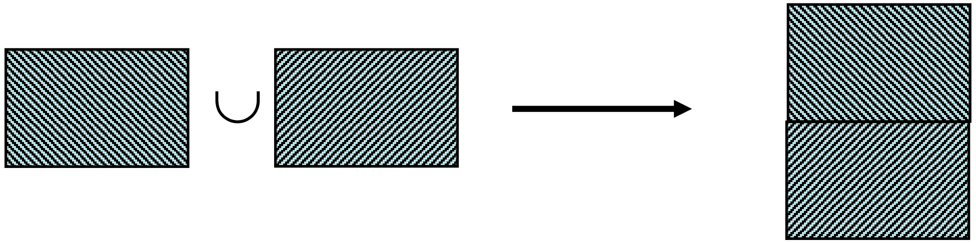
Projection



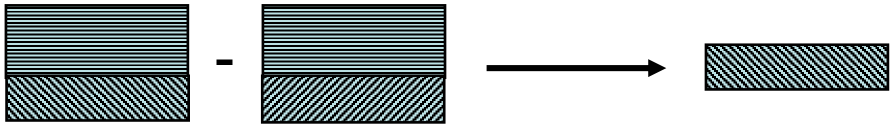
Selection



Union

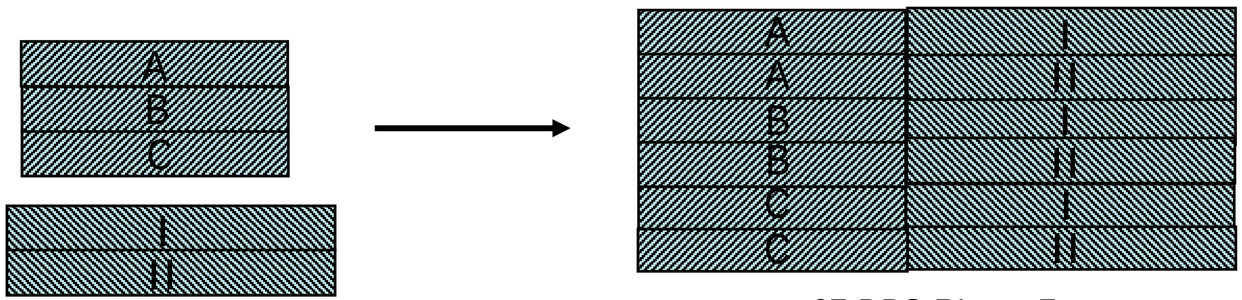


Difference



Cross Product

(Renaming)



Relational Algebra Operators

Freie Universität Berlin

{ Projection π,
(Extended) **Cross product** $\times$ **,**
Selection σ,
Union $\cup$ **,**
Set difference \ ,
Renaming ρ **}**

is a **base of relational operators.**
Other operators like **join (** $\bowtie$ **)** can be **expressed by** π,σ,ρ **,** $\times$**,** $\cup$

All operators can be expressed in SQL

5.2 Relational Algebra operations

Basic terminology (rep. from above)

Universal set of attributes A , $a_i \in A$ has domain $D(a_i)$

Relation Schema: named n-tuple of attributes
$RS = R(a1,...,an)$, $\{a1,\ldots an\} \subseteq A$

Schema operator Σ applied to relation R results in the type signature of R: $\Sigma(R) = R_A$

Relational **Database Schema**: set of relation schemas

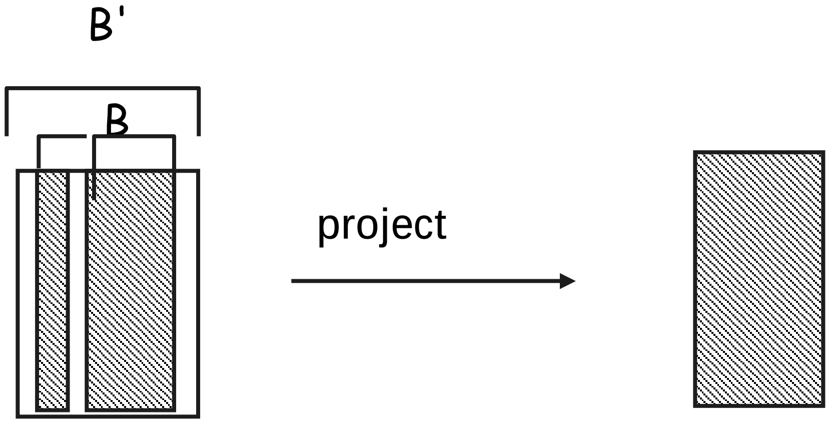
Database **Relation R**: subset of $D(a_{i1}) \times \ldots \times D(a_{in})$

Projection

Let $\Sigma(R) = B'$, $B \subseteq B'$

Projection π_B (R) of R on B:

Set of rows from R with the columns not in B eliminated



No duplicates in π_B (R) (in theory!)

Def.: π_B (R) = {r restricted to B | r $\in$ R}
= {r' | there is a tuple r $\in$ R such that
r' is the restriction of r to the attributes in B}

Projection (2)

Properties of projection:

- $|R| \geq |\pi_B(R)|, B \subseteq \Sigma(R)$
- B contains a key of R $\Rightarrow |\pi_B (R)| = |R|$

Useful for estimating the size of query results
Important for optimization.

SQL equivalent:
SELECT DISTINCT $b_1, b_2, \ldots b_n$ FROM R

Extended Cross Product X

Def.: (Extended) Cross product R X S =
{(a1,...an,b1,...,bm) | (a1,...,an) ∊ R, (b1,...,bm) ∊ S}

R (a1	a2)
1	'A'
5	'Z'

X

S (b1	b2)
3	'A'
1	'B'

=

T ((a1 ,	a2)	(b1	b2)))
(1	'A')	(3	'A')
(5	'Z')	(3	'A')
(1	'A')	(1	'B')
(5	'Z')	(1	'B')

SQL equivalent:
SELECT …
FROM R,S

SELECT …
FROM
R CROSS JOIN S

Renaming ρ

Def.: **Renaming**

Attributes: if $\Sigma(R) \cap \Sigma(S) \neq \emptyset$

ρ <attrname> ← <newAttrname> (<relname>)

Relations :

ρ <newname> (<relname>)

R (a1	a2)
1	'A'
5	'Z'

X

S (b1	a2)
3	'A'
1	'B'

$$\Sigma\,(\rho_{a2 \leftarrow b2}(S)) = \{b1, b2\}$$

Dot notation R.a2, S.a2 or explicit renaming

Renaming

$\rho_{\text{<newname>}}$ (<relname>)
Relation <relname> is renamed to <newname> in the context of expression

$\rho_{\text{<attrname>} \leftarrow \text{<newAttrname>}}$ (<relname>)
Attribute <attrname> of relation <relname> is renamed to <newAttrname> in the context of expression

```
π Sub.name (σ Q (σ P ( Employee X ( ρSub (Employee)))))
  where P = "Employee.name = 'Miller' "
        Q = "Sub.boss = Employee.id "
```

SELECT … FROM Employee, Employee **Sub**
WHERE …

Set operations

Freie Universität Berlin

r	(a1	a2)
	1	'B'
	5	'A'
	6	'B'

$\cup$

S	(a1	a2)
	'A'	2
	'B'	5

?

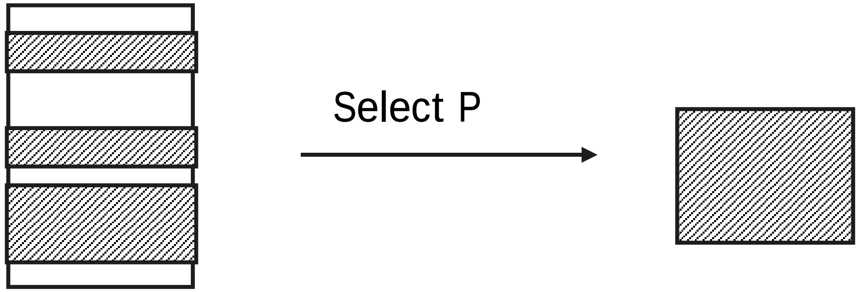
Def.: R and S are called **union-compatible** if the domains of Σ (R) = Σ (S)

R , S union-compatible,
then **set union** and **set difference**
R ∪ S and **R \ S** as defined on **mathematical sets**

SELECT …. FROM R …
{UNION | EXCEPT| INTERSECT}
SELECT … FROM S …

Selection σ

"Find cities with population more than 1 Mill .



Selection of tuples from a table R according to a predicate P defined on R

Def.: **Selection σ_P (R)**
Row predicate P:: R $\rightarrow$ {true, false}
$\sigma_P(R) = \{r \mid r \in R, P(r) = \text{true}\}$

Predicates

Row predicates:
inductively defined by primitive predicates
and boolean operators and, or not

Def.: Primitive (simple) predicates
Let a, b be attributes, w value from dom (a)
a θ **b** and **a** θ **w** are primitive predicates
where $\theta \in \{ =, !=, <, <=, >, >= \}$

Primitive predicates **compare**
either an attribute and a value
or two attributes

Row predicates

Syntax for **(row) predicates**

(i) Primitive predicates are predicates
(ii) If Q, Q' are predicates,
then $Q \wedge Q'$, $Q \vee Q'$ and $\neg (Q)$
are predicates
(iii) Operator preference and brackets as usual
(iv) There are no other predicates

"Find countries with more than 5 Mill population
and GNP <= 500

Selection of rows

Note:

Selection operator selects the row with all attributes:

$$\Sigma(R) \quad = \Sigma\ (\sigma_P\ (R)\ )$$

Size of result depends on **selectivity** of P

$$\textbf{selectivity} := |\ \sigma_P\ (R)\ |\ /\ |\ R\ |$$

important for optimization

SQL equivalent (but dupl.):
SELECT … FROM R
WHERE <row predicate P >

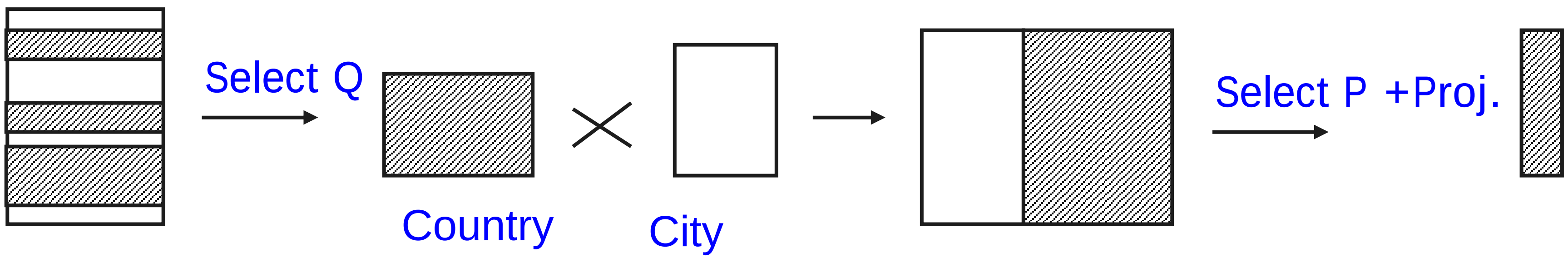
Note:
SQL block allows to combine

$\pi, \times, \sigma$

Combining operators

Country(C_id,name,...,population, ..)

'Find Countries which only consist of its capital and the population > 10000'
(Monaco is an example, Vatican not)



$$\pi_{name} (\sigma_P (\sigma_Q (\text{country} \times \text{City}))$$

where Q = "population > 10000 "

P = "capital = City.name $\wedge$

Country.pop = City.pop $\wedge$.. $\wedge$

```
SELECT Country.name FROM Country, City
WHERE C_id = City.C_id
        and capital = City.name   and R_id = City.R_id
        and Country.population = City.population
        and Country.population > 10000
```

SQL99 syntax

```
SELECT Country.name
FROM Country JOIN City ON C_id = City.C_id
                          and  capital = City.name
                          and R_id = City.R_id

WHERE Country.population = City.population
AND Country.population > 10000
```

Syntax of (simple) Relational Algebra defined inductively :

(1) Each table identifier is a RA expression

(2) ρ_A (B), $\rho_{s \leftarrow y}$ (A) are RA expressions where A,B table identifiers, s, v attribute identifiers

(3) If E and F are RA expressions then

π_D (E), σ_P (E), E X F, E $\cup$ F, E \ F are RA expressions (if union-compatible etc.)

where D $\subseteq$ Σ(E)

(4) These are all RA expressions

Semantics of Relational Algebra

val is a function which assigns to each relational algebra expression a result table:

val ('R') = R

"The value of a relation name is the relation (table)"

val ('τ (E)') = τ (val (E))

where τ is some unary rel. Operation like π

"The value of an unary relational operator applied to an relational algebra expression E is the result of applying the operator to the value of E "

val ('E ω F') = val (E) ω val (F)

where ω is some binary operator like X

"The value of an unary relational operator applied to a relational algebra expression E is the result of applying the operator to the value of E"

Remarks on RA and SQL

- Rewrite rule

  $$\sigma_{Q \wedge P}(R) = \sigma_Q(\sigma_P(R))$$

  implicitly used for SQL expression:

  `SELECT… FROM .. WHERE P (WHERE Q))`

  does not conform to SQL syntax

- **RA** results are **sets** (relations),
  SQL results are **bags** (duplicates allowed)

To eliminate duplicates write:

```
SELECT DISTINCT … FROM…
WHERE …P AND Q …
```

Renaming

Freie Universität Berlin

Renaming , why?

Example: **`Employee( id, name, boss, …)`**

`Find subordinates of 'Miller'`

~~π_{name} (σ$_P$ (σ$_Q$ (Employee) X Employee)))~~
~~where P = "Employee.name = 'Miller' "~~
~~Q = "Employee.boss = Employee.id "~~

RA is a **declarative** language: a **name denotes** the same relation / attribute within one expression

Evalution example: one table – two roles

Employee		
id	name	boss
001	Abel	NULL
002	Bebel	005
004	Cebel	005
005	Miller	001
006	Debel	001
		....

×

Sub		
id	name	boss
001	Abel	NULL
002	Bebel	005
004	Cebel	005
005	Miller	001
006	Debel	001
		....

ρ_{Sub} (Employee)

Renaming

Employee			Sub		
id	name	boss	id	name	boss
001	Abel	NULL	001	Abel	NULL
001	Abel	NULL	002	Bebel	004
...			...	...	
002	Bebel	005	001	Abel	NULL
002	Bebel	005	002	Bebel	005
...	...	...	...	...	...
005	Miller	001	001	Abel	Null
005	Miller	001	002	Bebel	005
005	Miller	001	004	Cebel	005
005	Miller	001	005	Miller	001
005	Miller	001	006	Debel	001
..	...	...	...	...	...
006	Debel	001	005	Miller	001
006	Debel	001	006	Debel	001

π_{name}

σ_P

σ_Q

5.4 Relational Algebra: More Operators

Some operation sequences occur frequently

⇨ define compound operators

Def.: Join (θ-join)

R, S relations, $R \bowtie_P S$

$= \{(a_1, ...a_n, b_1,...b_m) \mid P(a_1,...a_n,b_1,...b_m) \text{ is } \mathbf{true}\}$

$= \sigma_P (R \times S)$

where P is a (boolean) predicate composed of primitive predicates of the form

$a \; \theta \; b$, $a \in \Sigma(R)$, $b \in \Sigma(S)$, $\theta \in \{ =, \neq, <, <=, > >=\}$

(P is the join predicate)

Relational Algebra Join

$R \bowtie_{R.a < S.c \wedge R.b=S.d} S$ =

1	A	2	1	3	A
2	A	2	1	3	A

The result usually does not have a name

R(a b c)

1	A	2
2	A	2
3	C	1

S(a c d)

1	3	A
2	2	B
1	2	C

Note: exactly the same as taking the set of all pairs of R and S rows and checking the predicate subsequently

R X S

1	A	2	1	3	A
1	A	2	2	2	B
1	A	2	1	2	C
2	A	2	1	3	A
2	A	2	2	2	B
2	A	2	1	2	C
3	C	1	1	3	A
3	C	1	2	2	B
3	C	1	1	2	C

```
SELECT …
FROM R JOIN S on (R.a<S.d)
       AND  (R.b = S.d)
WHERE …
```

Relational Algebra : more operators

Equijoin: equality comparison

- **Most important type of join**: all primitive predicates in P compare **equality of column values** of two rows at a time : $P \equiv \wedge R.x_i = S.y_i$, $\{x_i\} \subseteq \Sigma(R)$, $\{y_i\} \subseteq \Sigma(S)$,
- Implements the "values as pointers" concept of RDB for foreign keys, but is more general.

Example using foreign key: Find Country name title of region having R_id = 'VAR'

$$\pi_{name} (Country \bowtie_{c\_id=c\_id} \sigma_{R\_id='VAR'} (Region))$$

Relational Algebra: renaming attributes

– Renaming **required**, if **identical column names**

– **No canonical projection** of columns if columns are redundant

R (x, y, z)

1	a	11
5	b	12
6	a	12

S(x', y, z')

7	a	23
6	c	15
9	a	3

$(R \bowtie_{R.y = S.y} S) =$

R (x, y, z , x' , y, z')

1	a	11	7	a	23
1	a	11	9	a	3
6	a	12	7	a	23
6	a	12	9	a	3

Relational Algebra: Natural join

Def.: Natural Join R ⋈ S :
equijoin over all **literally identical column names** of R and S and **projection of redundant columns**. Join predicate implicit.

R(a b c)

1	A	2
2	A	2
3	C	1

S(a c d)

1	3	A
2	2	B
1	2	C

R ⋈ S =

R.a	R.b	R.c	S.d
2	A	2	B
1	A	2	C

$$R \bowtie S = \pi_{\Sigma(R) \cup \Sigma(S)} (\sigma_P (R \times S))$$

where $P \equiv \bigwedge R.x = S.x, \quad x \in \Sigma(R) \cap \Sigma(S)$

SELECT … FROM R NATURAL JOIN S

Relational algebra: outer join

Motivation: only tuples of S participate in a join
R $\bowtie$ S, which have a "counterpart" in R.

Customer(<u>c_no</u>,name,f_name, zip, city)
Phones (<u>phoneNo</u>, <u>c_no</u>)

"Print telephon list of customers"

$\pi_{\text{name, phoneNo}}$ **(Customer $\bowtie$ Phones)**

Customers without phoneNo will not appear

Relational Algebra: outer join

Freie Universität Berlin

Left outer join $R \;⟕_P\; S$

Includes (r, NULL,…NULL) – if there is no join partner for $r \in R$

a	b	c
1	A	2
2	A	2
3	C	1

$⟕_{R.a < S.c \wedge R.b=S.d}$

a	c	d
1	3	A
2	2	B
1	2	C

=

1	A	2	1	3	A
2	A	2	1	3	A
3	C	1	-	-	-

Def.: $R \;⟕_P\; S =$

$R \bowtie_P S \cup \{ (r_1,\dots r_n, \text{NULL},\dots\text{NULL}) \mid (r_1,\dots r_n) \in R$ and for all $(s_1,..,s_m) \in S$: $P(r_1,\dots r_n, s_1,..s_m) = \text{FALSE} \}$

Outer join typically extension of equijoin

Relational Algebra: outer join

Freie Universität Berlin

Right outer join R ⟖ S

Includes (NULL,…NULL, s) – if there is no join partner for s $\in$ S

a	b	c
1	A	2
2	A	2
3	C	1

⟖ R.a < S.c $\wedge$ R.b=S.d

a	c	d
1	3	A
2	2	B
1	2	C

=

1	A	2	1	3	A
2	A	2	1	3	A
-	-	-	2	2	B
-	-	-	1	2	C

Full outer join: union of left and right outer join

1	A	2
2	A	2
3	C	1

⟗ R.a < S.c $\wedge$ R.b=S.d

1	3	A
2	2	B
1	2	C

=

1	A	2	1	3	A
2	A	2	1	3	A
3	C	1	-	-	-
-	-	-	2	2	B
-	-	-	1	2	C

Relational Algebra: More operators

Freie Universität Berlin

Def.: Semjoin

$$R \ltimes_P S = \Pi_{\Sigma(R)} (R \bowtie_P S)$$

Left semijoin is the subset of R, each r of which has a corresponding tuple s from S in the join.

Typically extension of equijoin or natural join

R(a b c) $\ltimes_{R.a = S.c \wedge R.b=S.d}$ S(a c d) = (a b c)

R(a b c):

a	b	c
1	**A**	**2**
2	A	2
3	C	1

S(a c d):

a	c	d
1	**3**	**A**
2	2	B
1	2	C

Result (a b c):

a	b	c
1	**A**	**2**

Right Semijoin defined symmetrically :

$$R \rtimes S = \Pi_{\Sigma(S)} (R \bowtie S)$$

Relational Algebra: Base operators

Base

Set of operators which allow to express all other operators

Relational operators
π, σ, $\times$, \ and $\cup$ form a **basis of relational algebra** operators

Means: every RA expression may be expressed only with these operators

Example: $R \bowtie_P S = \sigma_P (R \times S)$

Some rewrite rules for RA

Properties of selection and projection

$\sigma_P(\sigma_Q(R)) = \sigma_Q(\sigma_P(R))$

$\sigma_P(\sigma_P(R)) = \sigma_P(R)$

$\sigma_{Q \wedge P}(R) = \sigma_Q(\sigma_P(R)) = \sigma_Q(R) \cap \sigma_P(R)$

$\sigma_{Q \vee P}(R) = \sigma_Q(R) \cup \sigma_P(R)$

$\sigma_{\neg P}(R) = R \setminus \sigma_P(R)$, if $P(r)$ defined for all r (no NULL!)

if $X \subseteq Y \subseteq \Sigma(R)$ then $\pi_X(\pi_Y(R)) = \pi_X(R)$

if $X, Y \subseteq \Sigma(R)$ then $\pi_X(\pi_Y(R)) = \pi_{X \cap Y}(R) = \pi_Y(\pi_X(R))$

$attr(P) \subseteq X \subseteq \Sigma(R)$ then $\pi_X(\sigma_P(R)) = \sigma_P(\pi_X(R))$

where attr(P) denotes the set of attributes occuring in P

Relational Algebra operator trees

Algebraic Optimization

- Evaluation of RA expressions in canonical form

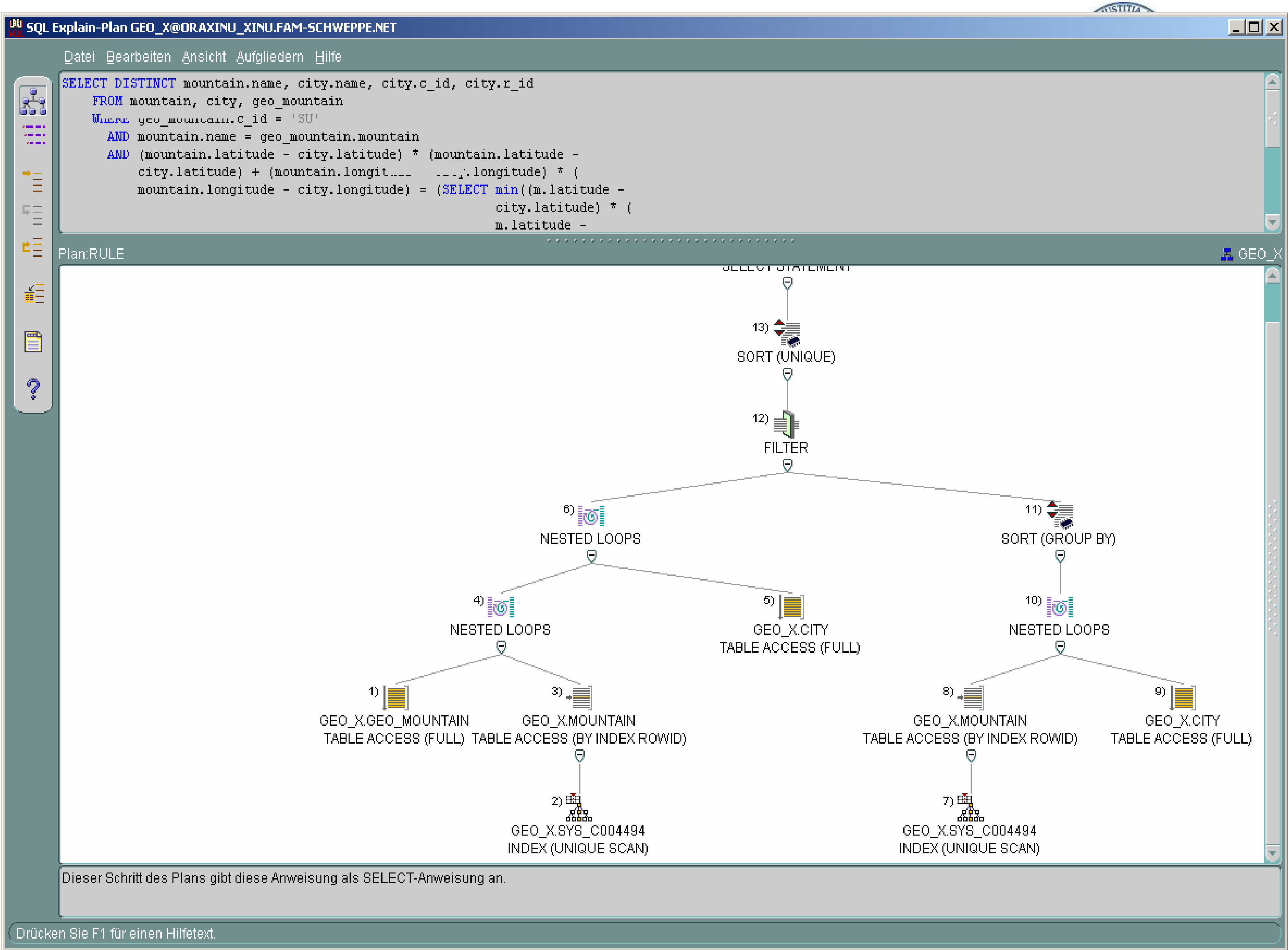
  π .... (σ_P ($R_1 \times R_2 \times \ldots \times R_n$))

  is very inefficient
- How to speed up evaluation of RA (and SQL) expressions?
- Example: Two tables R and S with n and m tuples
  Worst case complexity of :

  $\sigma_P (R \bowtie S)$

  is O(m*n)
- Interchange of select and join may result in O(n+m)

  time $\sigma_P (R) \bowtie S$ depending on the join algorithm


SQL Explain-Plan GEO_X@ORAXINU_XINU.FAM-SCHWEPPE.NET
Datei Bearbeiten Ansicht Aufgliedern Hilfe
SELECT DISTINCT mountain.name, city.name, city.c_id, city.r_id
FROM mountain, city, geo_mountain
AND mountain.name = geo_mountain.mountain
AND (mountain.latitude - city.latitude) * (mountain.latitude -
mountain.longitude - city.longitude) = (SELECT min((m.latitude -
city.latitude) * (
m.latitude -
Plan:RULE
GEO_X
13) SORT (UNIQUE)
12) FILTER
6) NESTED LOOPS
11) SORT (GROUP BY)
4) NESTED LOOPS
5) GEO_X.CITY TABLE ACCESS (FULL)
10) NESTED LOOPS
1) GEO_X.GEO_MOUNTAIN TABLE ACCESS (FULL)
3) GEO_X.MOUNTAIN TABLE ACCESS (BY INDEX ROWID)
8) GEO_X.MOUNTAIN TABLE ACCESS (BY INDEX ROWID)
9) GEO_X.CITY TABLE ACCESS (FULL)
2) GEO_X.SYS_C004494 INDEX (UNIQUE SCAN)
7) GEO_X.SYS_C004494 INDEX (UNIQUE SCAN)
Dieser Schritt des Plans gibt diese Anweisung als SELECT-Anweisung an.
Drücken Sie F1 für einen Hilfetext.

Relational Algebra: table predicates

Freie Universität Berlin

Row predicates:

P defined over rows (or pairs of rows)

Table predicates

Example: find all countries which are neighbors of all european Countries with population more than 78 Mill

Cannot be answered by comparing individual rows

Predicates with universal quantifier are table predicates

e.g. Find y_0 such that P(x) is true:

$$P(x) \equiv \forall x \; (PopGT70MillEurope\ (x) \Rightarrow \; (Q(x,y_0,)))$$

$$Q(x,y) \equiv x \text{ is neighbor of } y$$

- Express table predicates with base operators?

Relational Algebra: Division

Freie Universität Berlin

`Course(id,title,semester)`

`T ≡ Course_Stud(cid,matr#)`

ALP4	77
PSem	77
SW	55
SWT	12
SWT	77
ALP4	25
DBS	77
DBS	12

$F \equiv \pi_{id} (\sigma_{semester=B4}$ `Course(id,title, semester))`

ALP4	B4
DBS	B4
SWT	B4
PSem	B4

Result:

77

Find MatrNo of students who take all courses offered for semester B4.

Relational Division

Informally T . /. F is the set of all tuples r of T projected on attributes not belonging to F such that {(r)} X F $\subseteq$ T

Relational Division: example

T

ALP4	77
PSem	77
SW	55
SWT	12
SWT	77
ALP4	25
DBS	77
DBS	12

./.

F

ALP4
DBS
SWT
PSem

⇩

77
55
12
25

$\times$

ALP4
DBS
SWT
PSem

$\subseteq$

ALP4	77
PSem	77
SW	55
SWT	12
SWT	77
ALP4	25
DBS	77
DBS	12

$\Rightarrow (77) \in T./.F, \quad \{12,55,25\} \not\subset T./.F$

Relational Algebra: Division

Def.: Relational Division T . /. F

Attributes of F are a subset of the attributes of T:

- $\Sigma(F) \subset \Sigma(T)$
- Signature of T ./. F is $D = \Sigma(T) \setminus \Sigma(F)$

$$\mathbf{T\ ./.\ F} := \{ t' \mid t' \in \pi_D(T) \wedge (\forall s \in F)(\exists t \in T)\ \pi_{\Sigma(F)}(\{t\}) = s \wedge \pi_D(\{t\}) = t' \}$$

Simulates a finite "universal" quantification:
"For **all items x** in the table **holds** the predicate **P**"

Relational Algebra Division

Freie Universität Berlin

T ./. F may be defined in terms of other relational operators

$$T \;./.\; F = \pi_D(T) \setminus (\pi_D\;(\pi_D(T) \times F) \setminus T)$$

The "missing" tuples of T

Building the complement

D = Σ(T) \ Σ(F)

Proof: Assignment

Property of relational division:

Let D = Σ(T) \ Σ(F) ,

if D contains the key of T and |F| > 1 then T ./. F = ∅

5.5. Relational completeness

Completeness

- A DB language L is called **relational complete**, if every RA expression can be expressed in L
- Are there any operations on relations, which cannot be expressed by a finite RA expression (select, project, product or join; **SPJ**) ?
- Yes: **transitive closure** of a relation cannot be expressed in this way

Pred	Descend
Paul	Mary
Mary	Peter
John	Bill
Peter	George

No RA expression to find all decendents of 'Paul'.

Recursion is missing!

What is missing in RA

- Arithmetic operators,
- many practically important operators like **grouping** of results:
  "List Students and number of courses they take"

Matr#	NoOfCorses
77	4
55	1
12	2
25	1

ALP4	77
PSem	77
SW	55
SWT	12
SWT	77
ALP4	25
DBS	77
DBS	12

- More Predicates on tables (not rows)

Anyway **relational algebra important conceptual basis** for query languages and query evaluation

RA for optimization

An relational algebra **operator tree** is the data structure representing a RA expression

Algebraic optimization: systematic interchange of operation according to the laws of RA

Does not change time complexity in general, but "makes n small".

Implementation of Algebraic Optimization by transformation of the operator tree

- Systematic treatment of different optimization techniques → course "DB-Tech"

Summary

Relational algebra: **algebra on tables**

Operators: **project, select, cartesian product, union, set difference, (rename)**

Several compound operators : j**oin, outer join, semi-join, division**

Serves as a **basis for relational DB languages**

No recursion ⇨ not computationally complete

Base of **SQL**

Used for optimization by operator tree transformation

6. The Relational Data Model (*) : Logic foundation of data manipulation - in a nutshell -

Kemper / Eickler: Chap 3.5 , Elmasri/ Navathe: chap. 9.3+9.4
Garcia-Molina, Ullman, Widom: chap. 10,

(*) not discussed in class, not required for exam -

see also reader: logic&databases.pdf

6.1 Logical foundations of the RDM

Predicate logic (PL) view of a DB

Database may be seen as a set of facts:

- $r = (r_1,...,r_n) \in R$ for some relation R
- assign a predicate R' to R which is defined:

  $R'(r) = TRUE \iff r \in R$

  R' is a called a database predicate

Example:

Movie (25, Amistad, History, 1, Spielberg, 01.05.97) is a fact, "Movie" is a db predicate

RDM and predicate logic

Restrictions on PL formula

- only database predicates and comparison predicates (>, <, =, <=, >=, <>)
- Variables represent tuples (!)

Open and closed PL formula

- Closed : no free variables, i.e. every variable is bound by a quantifier.
  Example: see above
- Open: there are free variables, i.e not closed
- Example:

  ∃ t (Tape(t) ∧ t.movieId = m.mId ∧ t.format='DVD')
  Variable m is free in the formula

Open formula as queries

Open formula

$\exists$ t (Tape(t) $\wedge$ t.movieId = m.mId $\wedge$ t.format='DVD')

An open formula, the free (tuple) variable is m

$\exists$ m (Movie(m) $\wedge$ $\exists$ t (Tape(t) $\wedge$ t.movieId = m.mId $\wedge$ m.mId='4711' $\wedge$ t.format='DVD'))

and also

$\exists$ m (Movie(m) $\wedge$ $\exists$ t (Tape(t) $\wedge$ t.movieId = m.mId $\wedge$ t.format='DVD'))

are closed and can be evaluated to TRUE | False..

Tuple calculus

Interpret { (s.1) | P(x,y,…,s) } as:
all rows s which satisfy P(..)

s.1 means first component of (tuple) variable s

$\{ (s.1) \mid \exists\, m\ (Movie(m) \land \exists\, t\ (Tape(t) \land t.movieId = m.mId$
$\land\ s.1 = m.title \land t.format='DVD'))$

Formula is open because of s1

This is a declarative statement for the set given by the projection of all those s onto the first component, which make the predicate on the rifht hand side of | true.

Open formula as queries

- Implicit requirement: the database predicate of the variables must be known
  Technically speaking: the variables must be range coupled

Example

- $\{x.3 \mid Movie(x) \wedge x.title = P(...,x,...)\}$
- ' x is a variable, which represents tuples of Movie',
- Query result is the third component of those movie tuples which make P true.

6.2 Calculus Languages

Predicate logic as a query language

Called "calculus" languages for historical reasons

Two types of languages

Domain calculus

All **variables represent typed values** (or domains) of the relations of the DB (domain variables)

Tuple calculus

The **variables** in expressions **represent a row** (tuple) of a relation (tuple variable)

Tuple Calculus

Tuple Calculus language

- Defined over
  - predicates R, S, T,... which correspond to database relations
  - set A of attribute names { a, b,.. }
  - values from the domains of A
  - tuple variables r,s,t,...
- A tuple calculus expression (with one free variable) has the form

  {s | F(s)}

  where s is a tuple variable and F(s) a formula in which s occurs free

Example

$\{(s.1,s.2) \mid \exists\, m\ (Movie(m)\ \wedge\ m.title = s.1 \wedge$ and
$m.year > '1992' \wedge\ m.year = s.2)\}$

- i.e. all movie and year of production titles produced after 1999

Tuple calculus

Freie Universität Berlin

- s is called the target list
- s is not range coupled and its components are denoted s.1, s.2,….s.k, if there are k output components
- s.i has to be connected in F to some range coupled variables

Example:

{(s.1,s.2) | ∃ m (Movie(m)

∧ m.title = s.1 ∧ and m.year> '1992' ∧ m.year=s.2)}

i.e. all movie and year of production titles produced after 1999

Tuple calculus

Simplification

$\{(s.1,s.2) \mid \exists\, m\ (Movie(m)$
$\wedge\ m.title = s.1 \wedge$ and $m.year > '1992' \wedge m.year=s.2)\}$

Substitute "logical variables" s.i by row variables

$\{(m.title, m.year) \mid \exists\, m\ (Movie(m)\ \wedge$ and $m.year > '1992'\ )\}$

Eliminate existential quantifiers : "all free variables are existentially quantified" … at least those in target list.

$\{(m.title,m.year) \mid\ Movie(m) \wedge$ and $m.year > '1992'\ \}$

Range coupling of (row) variable m

Tuple calculus and relational algebra

Freie Universität Berlin

Tuple calculus expression for algebra operators

– Projection, cross product

$\pi_{a,b}(R \; X \; S) \quad \equiv$
$\{ (x.a, y.b) \mid R(x) \wedge S(y) \}$

– Join $R \bowtie_P S$

– $\equiv \{r.e,\ldots,t.k \mid R(r) \wedge S(t) \wedge P\}$

– Selection
e.g. $\sigma_{(s.a = v \vee s.a = w) \wedge s.b = s.c}(R)$
$\equiv \{(s.a,\ldots,s.k) \mid R(s) \wedge (s.a = v \vee s.a = w) \wedge s.b = s.c\}$

More examples in the class

Tuple calculus: examples(1)

Movies (title) all copies of which are on loan

$\{m.title \mid Movie(m) \wedge \forall\, t\, (Tape(t) \wedge m.m\_id = t.m\_Id$
$\Rightarrow \exists\, x\, (\, Rental(x) \wedge x.t\_id = t.t\_id))\, \}$

Find movie titles available in all formats (in the DB)

"… for all formats there exists a tape with this format and this movie"

$\{m.title \mid Movie(m) \wedge \forall\, f\, (Format(f) \;\Rightarrow\; \exists\, x\, (Tape(x) \wedge$
$f.format = x.format \wedge x.m\_id = m.m\_id))\}$

Examples (2)

"Find actors who played together in the same movie."

$\equiv$

'There exists an actor and another actor and two different "starring" entries, such that the movie-attributes of both entries are the same and the actor attribute values are the foreign key values for these two actors '

$$
\{(a1.stage\_name,\ a2.stage\_name) \mid Actor(a1) \wedge Actor(a2) \wedge \\
\exists s1\ (\ Starring(s1) \wedge \exists\ s2\ (\ Starring(s2) \wedge s1.actor\_name = \\
\exists a1.stage\_name\ \wedge\ s2.actor\_name = a2.stage\_name\ \wedge \\
\exists s1.movie\_id = s2.movie\_id\ and\ s1 <> s2\ \}
$$

Tuple calculus used in Ingres / UC Berkeley as data handling language QUEL. Successor Postgres : SQL

Limitations of RCalc and safe expressions

Limitations, extensions and issues

- Difference to first order predicate logic (FOL)
  - no functions $\forall$ x (x > 1 $\Rightarrow$ square(x) > x) not allowed
  - FOL interested in formula valid for all domains
    e.g. $\forall$ x P(x) $\vee \neg$ P(x)
  - RC: Interpretation of tuple calculus expressions over the DB
- What does $\{x \mid \neg R(x)\} = \{x \mid \neg \exists t\, (R(t) \wedge x = t\}$ mean?
- All tuples NOT belonging to R may not even be a finite set

Safeness of Relational Calculus expressions

A tuple calculus expression is called safe, if the result is finite

– Unfortunately safety property is **not decidable**

– Roughly speaking (syntactically), expressions are safe, if no range variable occurs negated outside an expression which restricts the result set otherwise

– e.g. $\{x \mid R(x)\}$ and $\{x \mid T(x) \wedge \neg R(x)\}$ are safe,

– but $\{x \mid \neg R(x)\}$ is NOT!

6.3 Relational completeness

Relational Algebra and calculus are equivalent

- For each **RA** expression there is an equivalent **safe tuple calculus** expression
- For each **safe tuple calculus** expression there is an **equivalent safe domain calculus** expression
- For each **safe domain calculus** expression there is an **equivalent RA** expression

  Equivalent means: results are the same when evaluated over the same DB

- This property of relational languages is called

  relational completeness

Relational complete does not mean computational complete.

Relational completeness

Has been considered as the base line for database query languages: every **query language** should be **as expressive as relational algebra**

SQL is in this tradition, but has introduced many concepts which are difficult or impossible to express in RA

- **grouping and predicates over sets**

  e.g. find those movies having the maximal number of copies (DVDs)
- **arithmetic in expressions**, e.g. find cheapest product prices including taxes (in an appropriate DB)
- **partial matches of attribute values**, e.g. find movies the titles of which are LIKE 'To be$ '
- **application specific comparison functions** (and types), e.g:

  find those customers whose names sound like "Maia"

Relational Algebra

- **Applicative language** on tables for specifying result tables
- **Base for SQL (partially)** and query optimization

Relational Calculus:

- Formal languages syntactically and sementically based on predicate claculus for handling data in relational model
- **Declarative language**, specify *which*, not *how*, data to retrieve
- **Base for** QUEL, QBE, **SQL (partially)**

l