Relational Algebra

- set of relational model operations to manipulate data
- enable the user to specify retrieval requests
- results are new relations, that can in turn be manipulated by the same operations

Relational Algebra Expression:

sequence of relational algebra operations

The result of a relational algebra expr. is again a relation

relational algebra operations $\nearrow$ set-theoretic operations

relational algebra operations $\searrow$ relational operations

The SELECT operation

select a subset of the tuples of a relation that satisfy a selection condition

(filter tuples that satisfy a condition)

Example: $\sigma_{DNO=4}(EMPLOYEE)$

Example: $\sigma_{SALARY>30000}(EMPLOYEE)$

Example: $\sigma_{(DNO=4\ AND\ SALARY>25000)\ OR\ (DNO=5\ AND\ SALARY>30000)}(EMPLOYEE)$

general syntax of the SELECT operation:

$\sigma_{condition}(R)$

- ***condition*** is a Boolean expression made up from clauses
  - <attrib name> <comparison operator> <value>
  - <attrib name> <comparison operator> <attrib name>
  - comparison operator is one of $=, \leq, \leq, \geq, \geq \neq$ except when the attribute domain is a set of unordered values (e.g. colors)
  - value is an element of the attribute domain
  - clauses can be combined with the boolean operators AND, OR, NOT
- R is a relation, in general a relational algebra expression

properties of the SELECT operation:

- Determination of the result of SELECT: apply the select condition to each tuple and evaluate the condition
- SELECT operation is unary
- degree($\sigma_{condition}(R)$) = degree(R)
- # of tuples of $\sigma_{condition}(R) \leq$ # of tuples of R
- selectivity of a condition: how many tuples are selected
- commutativity of the SELECT operation:
  $\sigma_{condition1}(\sigma_{condition2}(R)) = \sigma_{condition2}(\sigma_{condition1}(R))$
- nested SELECT operations can be replaced with a multiple AND condition:
  $\sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R)))) = \sigma_{c1\ AND\ c2\ AND\ \ldots\ AND\ c_n}(R)$

The PROJECT operation

project a relation wrt some attributes only

SELECT works on rows, PROJECT works on columns

`Example:` $\pi_{LNAME,FNAME,SALARY}(EMPLOYEE)$

`Example:` $\pi_{SEX,SALARY}(EMPLOYEE)$

general syntax/properties of the PROJECT operation:

$\pi_{attribute\ list}(R)$

- PROJECT operation is unary
- degree($\pi_{attribute\ list}(R)$) = # attributes in $attributelist$)
- # of tuples of $\pi_{attribute\ list}(R) \leq$ # of tuples of R
  (it can be less because of duplicate elimination)
  (equality occurs when the attribute list is a superkey of R)
- commutativity does not hold for the PROJECT operation
- $\pi_{l_1}(\pi_{l_2}(R)) = \pi_{l_1}(R)$ when l_2 contains the attributes of l_1

The RENAME operation & Sequences of Operations

Sometimes we want to be able to give names to intermediate results of relational algebra operations.

Thus we avoid complicated nested relational algebra expressions.

Example: $\pi_{FNAME,LNAME,SALARY}(\sigma_{DNO=5}(EMPLOYEE))$

can be decomposed as follows:

DEP5_EMP $\longleftarrow \sigma_{DNO=5}(EMPLOYEE)$

RESULT $\longleftarrow \pi_{FNAME,LNAME,SALARY}$(DEP5_EMP)

we can also rename the attributes of the result:

TMP $\longleftarrow \sigma_{DNO=5}(EMPLOYEE)$

R(FIRSTNAME, LASTNAME, SALARY) $\longleftarrow \pi_{FNAME,LNAME,SALARY}$(TMP)

general syntax of the RENAME operation:

suppose we have a relation R of degree n ,

$$\boldsymbol{R(A_1, A_2, \ldots, A_n)}$$

we can rename the relation, the attributes or both:

- $\boldsymbol{\rho_{S(B_1,B_2,\ldots,B_n)}(R)}$
- $\boldsymbol{\rho_S(R)}$
- $\boldsymbol{\rho_{(B_1,B_2,\ldots,B_n)}(R)}$

Set Theoretic Operations

Relations are sets and as such they are subject to the usual set-theoretic operations (merging sets).

$\texttt{DEP5\_EMPS} \longleftarrow \boldsymbol{\sigma}_{\boldsymbol{DNO=5}}(\texttt{EMPLOYEE})$

$\texttt{RES1} \longleftarrow \boldsymbol{\pi}_{\boldsymbol{SIN}}(\texttt{DEP5\_EMPS})$

$\texttt{RES2(SIN)} \longleftarrow \boldsymbol{\pi}_{\boldsymbol{SUPERSIN}}(\texttt{DEP5\_EMPS})$

$\texttt{RES} \longleftarrow \texttt{RES1} \cup \texttt{RES2}$

RES will contain the SINs of all employees who either work in dpt $\mathbf{5}$ or directly supervise an employee who works in dpt $\mathbf{5}$.

- The 3 set-theoretic operations, UNION, INTERSECTION & SET DIFFERENCE are binary

- To ensure the applicability of the 3 set-theoretic operations, the Union Compatibility condition must hold

- Union Compatibility:

$$R(A_1, A_2, \ldots, A_n), S(B_1, B_2, \ldots, B_n)$$

same degree n and $dom(A_i) = dom(B_i), i = 1, \ldots, n$.

- UNION $\boldsymbol{R \cup S}$ (commutative, duplicate elimination applies)
- INTERSECTION $\boldsymbol{R \cap S}$ (commutative)
- SET DIFFERENCE $\boldsymbol{R - S}$ (non commutative)
- Convention: the resulting relation has as attribute names: $\boldsymbol{A_1, A_2, \ldots, A_n}$
- UNION & INTERSECTION are associative:
  $\boldsymbol{R \cup (S \cup T) = (R \cup S) \cup T}$ $\qquad$ $\boldsymbol{R \cap (S \cap T) = (R \cap S) \cap T}$

CARTESIAN PRODUCT

- (alias CROSS PRODUCT or CROSS JOIN)
- binary operation, does not require Union Compatibility
- $R(A_1, A_2, \ldots, A_n) \times \boldsymbol{S(B_1, B_2, \ldots, B_m)} \Longrightarrow \boldsymbol{Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)}$
- degree(Q) = degree(R) + degree(S)
- # tuples in Q = (# tuples in R) $\star$ (# tuples in S)
- in general the result of $\boldsymbol{R} \times \boldsymbol{S}$ does not make sense
- useful when followed by a selection that matches values of attributes of $\boldsymbol{R}, \boldsymbol{S}$

Example: **retrieve the list of names of the dependents of each female employee**

$\texttt{FEMALE\_EMPS} \longleftarrow \boldsymbol{\sigma_{SEX=F}}(\texttt{EMPLOYEE})$

$\texttt{EMPNAMES} \longleftarrow \boldsymbol{\pi_{FNAME,LNAME,SIN}}(\texttt{FEMALE\_EMPS})$

$\texttt{EMP\_DEPENDENTS} \longleftarrow \texttt{EMPNAMES} \times \texttt{DEPENDENT}$

$\texttt{ACTUAL\_DEPENDENTS} \longleftarrow \boldsymbol{\sigma_{SIN=ESIN}}(\texttt{EMP\_DEPENDENTS})$

$\texttt{RES} \longleftarrow \boldsymbol{\pi_{FNAME,LNAME,DEPENDENT\_NAME}}(\texttt{ACTUAL\_DEPENDENTS})$

JOIN Operation

- denoted by $\bowtie$
- used to combine related tuples from 2 relations into single tuples
- used to express relationships between relations

Example: **JOIN**

Retrieve the name of the manager of each department.

We need to combine each DEPARTMENT tuple with the EMPLOYEE tuple whose SIN value matches the MGRSIN value in the DEPARTMENT tuple.

DEPT_MGR $\leftarrow$ DEPARTMENT $\bowtie_{MGRSIN=SIN}$ EMPLOYEE

RES $\leftarrow$ $\pi_{DNAME,FNAME,LNAME}$(DEPT_MGR)

MGRSIN is a foreign key, referential integrity. Do this example with Cartesian Product.

general syntax: $R \bowtie_{join\ condition} S$

$$R(A_1, \ldots, A_n) \bowtie S(B_1, \ldots, B_m) \Rightarrow Q(A_1, \ldots, A_n, B_1, \ldots, B_m)$$

degree $R \bowtie_{join\ condition} S$ = degree R + degree Q $= n + m$

the result of the join contains each combination of tuples (one from R and one from S) whenever this combination satisfies the $join\ condition$.

(difference of join and cartesian product)

the join condition is specified on attributes of the two relations R and S and is evaluated for each combination of tuples.

each combination for which the join condition evaluates to true, is included in the result Q.

Nested JOINs:

$$((PROJECT \bowtie_{DNUM=DNUMBER} DEPARTMENT) \bowtie_{MGRSIN=SIN} EMPLOYEE)$$

- general *join condition*:

$$< condition > AND \ldots AND < CONDITION >$$

where each *condition* is of the form $A_i\ \theta\ B_j$ with A_i an attribute of R, B_j an attribute of S, θ is one of the comparison operators $\{=, \neq, <, \leq, >, \geq\}$.

Note 1: A_i and B_j must have the same domain.

Note 2: Tuples whose join attribute values are NULL, do not appear in the result.

Types of JOINs:

- **THETA JOIN** the join condition has the general form described above
- **EQUIJOIN** JOIN in which the only comparison operator used is $=$
- **NATURAL JOIN** [denoted by $\star$] EQUIJOIN followed by removal of the redundant (same values) attribute(s)

Example: **NATURAL JOIN**

PROJ_DEPT ←
PROJECT ⋆ $\rho_{DNAME,DNUM,MGRSIN,MGRSTARTDATE}$(DEPARTMENT) **Renaming is used to achieve a common name ($DNUM$) for the JOIN ATTRIBUTE.**

Example: **NATURAL JOIN**

DEPT_LOCS ← DEPARTMENT ⋆ DEPT_LOCATIONS
The common join attribute $DNUMBER$ is omitted from the notation.

- **If no tuples satisfy the join condition, the result is an empty relation.**
- **If all tuples satisfy the join condition, the result is the cartesian product.**
- **# tuples in $R \bowtie_c S$ $\leq$ # tuples in R · # tuples in S.**

Join Selectivity: ratio of tuples in the result over the maximum number of tuples.

Complete Set of RA operations

Fact:

The set of RA operations $\{\sigma, \pi, \cup, -, \times\}$ is a **complete** set.

- INTERSECTION:
  $R \cap S = (R \cup S) - ((R - S) \cup (S - R))$
- JOIN: $R \bowtie_c S = \sigma_c(R \times S)$

DIVISION Operation

- denoted by $\div$
- useful for retrieval requests that contain a universal quantifier ($\forall$)
- binary operation, defined as follows:

  $R(Z) \div S(X) = T(Y)$ where $X \subset Z$ and where $Y = Z - X$ (Y denotes the set of attributes of R that are not attributes of S)

  a tuple t appears in the result $T(Y)$ if the Y value(s) of t appears in R combined with **every** X value(s) of S.
- expression of DIVISION via the complete set of RA operations:

$$R(Z) \div S(X) = \pi_Y(R) - \pi_Y((S \times \pi_Y(R)) - R)$$

Example: **DIVISION (1st abstract example)**

$R \div S = T$

R	A	B
	a_1	b_1
	a_2	b_1
	a_3	b_1
	a_4	b_1
	a_1	b_2
	a_3	b_2
	a_2	b_3
	a_3	b_3
	a_4	b_3
	a_1	b_4
	a_2	b_4
	a_3	b_4

S	A
	a_1
	a_2
	a_3

T	B
	b_1
	b_4

Which tuple(s) must we add to R so that b_2 belongs to the result T?

Which tuple(s) must we take out from R so that b_4 does not belong to the result T?

Division again

Another point of view:
Analogy with integer division.

$7/3 = 2$ because $2 \cdot 3 \leq 7$

A/B is the maximum integer Q s.t. $\quad Q \cdot B \leq A$

$A \div B$ is the largest relation Q s.t. $\; Q \times B \subseteq A$

`Example:` **DIVISION (2nd abstract example)**

A	S	R
	s_1	r_1
	s_1	r_2
	s_1	r_3
	s_1	r_4
	s_2	r_1
	s_2	r_2
	s_3	r_2
	s_4	r_2
	s_4	r_4

B1	R
	r_2

B2	R
	r_2
	r_4

B3	R
	r_1
	r_2
	r_4

Compute $A \div B1$, $A \div B2$, $A \div B3$.

Other technique to compute division: **Compute all s-values of A that are not disqualified. An s value s_i is disqualified if by attaching an r-value r_j from B, we obtain $(s_i, r_j) \notin A$.**

Example: **DIVISION (COMPANY example)**

Retrieve the name of employees who work on all the projects that "John Smith" works on. *Separate the query into 3 parts:*

(a) Build the list of PNOs that JS works on:

SMITH $\longleftarrow \sigma_{FNAME='John' AND\ LNAME='Smith'}(EMPL)$
SMITH_PNOS $\longleftarrow \pi_{PNO}(WORKS\_ON \bowtie_{ESIN=SIN} SMITH)$

(b) Build a table that contains tuples $< PNO, ESIN >$

SIN_PNOS $\longleftarrow \pi_{ESIN,PNO}(WORKS\_ON)$

(c) Apply DIVISION to get the result:
SINS(SIN) $\longleftarrow SIN\_PNOS \div SMITH\_PNOS$
RESULT $\longrightarrow \pi_{FNAME,LNAME}(SINS \star EMPLOYEE)$

Aggregate Functions & Grouping

- specify mathematical **aggregate functions** on collections of values from the database:

`SUM, AVERAGE, MAXIMUM, MINIMUM`

- `COUNT` is used to count tuples or values
- group the tuples of a relation according to the values of a certain attribute and then apply an aggregate function to each group separately

syntax: $_{<grouping\ attributes>}\mathcal{F}_{<function\ list>}(R)$

where $< grouping\ attributes >$ is a list of attributes of R and $< function\ list >$ is a list of pairs of the form $(< function >, < attribute >)$ where $< function >$ is one of SUM, AVERAGE, MAX, MIN, COUNT

The resulting relation has the grouping attributes plus one attribute for each pair in the $< function\ list >$

Example: **AGGREGATION & GROUPING**

For each department retrieve its DNO, its # employees and their average salary.

$$RES \leftarrow {}_{DNO}\mathcal{F}_{COUNT\ SIN,AVERAGE\ SALARY}(EMPL)$$

using renaming: $\rho_{R(DNO,NO\_OF\_EMPLS,AVG\_SAL)}(RES)$

Example: **AGGREGATION & GROUPING**

If no renaming occurs, the attributes of the resulting relation are named by concatenating the name of the function and the attribute in each pair of the $< function\ list >$

$${}_{DNO}\mathcal{F}_{COUNT\ SIN,AVERAGE\ SALARY}(EMPLOYEE)$$

Example: **AGGREGATION & GROUPING**

If no grouping attributes are specified, the functions are applied to all the tuples in the relation, so there will be only one tuple in the resulting relation

$$\mathcal{F}_{COUNT\ SIN,AVERAGE\ SALARY}(EMPLOYEE)$$

When aggregate functions are applied, duplicate elimination is not performed, in order to compute SUM and AVERAGE accurately.

Recursive Closure

operation applied to recursive relationships, like **supervision**

what are the employees supervised by a certain employee?

what are the employees supervised by the employees supervised by a certain employee?

we can specify an arbitrary level of nesting in these types of queries using successive join operations.