Database Design & Applications (SS ZG 518)

BITS Pilani
Hyderabad Campus

Dr.R.Gururaj
CS&IS Dept.

Relational Algebra & Relational Calculus

innovate achieve lead

Content

- *Query languages & Formal query languages for Relational data model*
- *Introduction to Relational Algebra*
- *Relational operators*
- *Set operators*
- *Join operators*
- *Aggregate functions*
- *Grouping operator*
- *Relational Calculus concepts*

Query Languages for Relational data model

Querying means extracting data from the database for the purpose of processing it.

Every data model has some *formal query languages* to support specification of data retrieval and manipulate requests.

Formal query languages

1. *Relational Algebra*
2. *Relational Calculus*
   - *(a) Tuple Relational Calculus*
   - *(b) Domain Relational Calculus*

Commercial query languages

1. *Structured Query Language (SQL)*
2. *Query by Example (QBE)*

Introduction to Relational Algebra

Relational Algebra is a formal query language for relational data model.

A basic set of relational model operations constitute the relational algebra.
These operations enable the user to specify basic data retrieval requests.
The result of a relational algebra query is also a new relation which may have been formed from one or more relations.

A sequence of relational algebraic operations forms a *relational algebraic expression*, whose result is also a relation.

Operations in Relational Algebra

innovate achieve lead

A. ***Set Operations***

- Union,
- Intersection,
- Difference,
- Cartesian product.

B. ***Relational Operations***

- Select,
- Project,
- join,
- Division etc.

Select Operation: is to select subset of tuples that satisfy some selection condition.
Symbol used is σ *(sigma)*

Ex: $\sigma_{dno=4}(\text{EMP})$

The above expression selects all tuples from EMP table, where the value of the column '*dno*' is 4.

The general form of '*select*' clause is $\sigma_{\text{<select condition>}}(R)$

Projection Operation:
Selects certain columns. Symbol is π*(pi)*

$\pi_{\text{name, age, dno}}(\text{EMP})$

Selects columns *name, age, dno* for all tuples from the table EMP

Note:

We can apply the expressions in sequence or we can nest them in single expression.

Ex.: $\pi_{\text{name, age}}\left(\sigma_{\text{dno=5}}(EMP)\right)$

The above expression selects *name* and *age* of employees working with *dno* 5.

The above query can also be written as

$$R_1 \leftarrow \sigma_{\text{dno=5}}(EMP)$$

$$R_2 \leftarrow \pi_{\text{name,age}}(R_1)$$

R_1 & R_2 are the names given to intermediate results(relations).

Union:

If two relations R_1 & R_2 are compatible (i.e., have same type of tuples) then we can merge them by union operation.

Duplicate tuples are eliminated. Ex: $(R_1 \cup R_2)$.

Intersection $R_1 \cap R_2$

Only equivalent tuples from R_1 & R_2 are selected.

Difference $R_1 - R_2$

Only those tuples seen in R_1 and not seen in R_2 are selected.

Note: (R1-R2) is not same as (R2-R1)

$$R_1 \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

no rows = 3

no Columns -2

Rows=3

Column=2

$$R_2 \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

$(R_1 \times R_2) =$

$$\begin{bmatrix} a_{11} & a_{12} & b_{11} & b_{12} \\ a_{11} & a_{12} & b_{21} & b_{22} \\ a_{11} & a_{12} & b_{31} & b_{32} \\ a_{21} & a_{22} & b_{11} & b_{12} \\ a_{21} & a_{22} & b_{21} & b_{22} \\ a_{21} & a_{22} & b_{31} & b_{32} \\ a_{31} & a_{32} & b_{11} & b_{12} \\ a_{31} & a_{32} & b_{21} & b_{22} \\ a_{31} & a_{32} & b_{31} & b_{32} \end{bmatrix}$$

No of rows = $3 \times 3 = 9$

No Columns = $2 + 2 = 4$

Cross product or Cartesian product

Rename operator

ρ*(rho)*

Ex: $\rho_{S(b_1,b_2,b_3)}(R)$

Renames R to S and new names of attributes are b_1, b_2, b_3

$\rho_S(R)$

Renames R to S with same attribute names

Division (÷)

Used when we want to check the meeting of all the criteria

Let R(A, B) and S(A) $T \leftarrow R \div S$

Selects all values for *B* column in R which contains all values under A in S.

Hence the no. column in T is only *B*.

Join: ($\bowtie_{\theta}$) Used to join tuples from different tables based on same condition. Result is new tuple with different arity.

$D \leftarrow DEPT \bowtie_{Mgrssn = ssn} EMP$

Joins tuples from DEPT & EMP where *Mgrssn* in DEPT is equal to *ssn* in EMP and stores the new tuples in relation D.

Division operator

$T = R \div S$

R

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

S

B
b1
b2
b3

T

A
a1
a4

Theta Join: $\bowtie_{\theta}$
Joining tables on some condition.

Natural Join: is joining of tables on attributes in R and S having the same label.

In the resulting relation only one column is listed.

Ex. D $\leftarrow$ DEPT $\bowtie$ EMP.

The joining is on common attribute with same name (*Dept Name*).

Employee				Dept			Employee * Dept			
Name	**EmpID**	**Dept Name**		**Dept Name**	**Manager**		**Name**	**EmpID**	**Dept Name**	**Manager**
Harry	3415	Finance		Finance	George		Harry	3415	Finance	George
Sally	2241	Sales		Sales	Harriet		Sally	2241	Sales	Harriet
George	3401	Finance					George	3401	Finance	George
Harriet	2202	Sales		Production	Charles		Hariet	2202	Sales	Harriet

Inner Join (R ⋈ S) An inner join only combines tuples from R and S if they meet the conditions. Tuples that do not meet the conditions are not showed in the final result. (This is the usual type of join).

Outer join: An outer join displays the tuples of one of the relations even if there is no match for the tuple in the other relation.

Left outer join: (R ⟕ S) In the result relations, in addition to all the matching tuples from R and S, all the remaining tuples from left side relation (R) are also showed. For these tuple from R, columns under S will have null values(padding).

Right Outer Join: (R ⟖ S) In the result relation, matching tuples will occur from R & S. In addition all the tuples form S will also appear with null values for the R attributes.

Full Outer join (R ⟗ S) In the result, all tuples from R & S will appear with null values for the other relations attributes.

Additional Relational Operations

Aggregate functions: Sum Average Max Min Count

Grouping:

The tuples of a relation are first grouped by the value of some attribute and then aggregate functions are applied on individual groups.

Symbol used is – £

Ex. $Dno\ £_{Count\ (ssn)}\ (EMP)$

The above expression first group the tuples in EMP table based on Dno, and then applies count function on individual groups this will output no. of employees in each department.

Result relation →

Dno	Count (SSn)

Examples

Look at the following relational schemas.

Vendor (<u>vid</u>, vname, city, category)
Part (<u>pid</u>, pname, price)
Supply (<u>pid</u>, <u>vid, qty</u>)

Vendor

vid	vname	city	cat
v1	Goel	Delhi	A
v3	Hameed	Kolkata	C
v6	Khanna	Delhi	C

Supply

pid	vid	qty
p1	v1	2
p1	v3	7
p5	v1	4
p5	v3	1
p5	v6	6

Part

pid	pname	price
p1	Wheel	500
p2	Shaft	200
p5	Injector	600
p9	bolt	100

Get the partID and part name for those parts supplied by all vendors from 'KOLKATA'.

$$R_1 \leftarrow \Pi_{vid}\left(\sigma_{city='kolkata'}(Vendor)\right)$$

$$R_2 \leftarrow \Pi_{pid,vid}(Supply)$$

$$R_4 \leftarrow (R_2 \div R_1)$$

$$Result \leftarrow \Pi_{pid,pname}(R_4 * Part)$$

For each ‘A’ category vendor, give vendor ID, vendor name and number parts that vendor has supplied.

$$R_1 \leftarrow \Pi_{Vid}\left(\sigma_{Cat='A'}(Vendor)\right)$$

$$R_2 \leftarrow Vid\,\mathcal{F}_{Count(Pid)}(R_1 * Supply)$$

$$Result \leftarrow \Pi_{Vid, Vname, Count\_Pid}(Vendor * R_2)$$

Examples for data retrieval using RA

1. Get the partID and part name for those parts not supplied by any vendors from ‘Delhi’.
2. Give vendorID and name who are not supplying any part.
3. Give partID and name which are supplied by at least 5 vendors.
4. Give vendorID and name who supply all parts.
5. For each part give part ID name and total quantity supplied by all vendors together.
6. Give vendorId and name for those who supply at least one part with cost more than 400.
7. Get PartId and name for those parts not supplied by any vendor from ‘Mumbai’

Tuple Relational Calculus

Relational Calculus is a formal query language for relational model where we write one declarative expression to specify a retrieval request and hence there is no description of how to evaluate the query.

A calculus expression specifies what is to be retrieved rather than how to do it.

Hence, relational calculus is *non–procedural* language where as *relational algebra* discussed in the previous section is procedural, where we write sequence of operations to retrieve data.

Any expression for data retrieval written in relational algebra can also be written in relational calculus and vice-versa.

Hence expression power of relational algebra and relational calculus is same.

Tuple Rational Calculus(TRC) is based on specifying a number of *tuple variables*.

Each tuple variable usually ranges over a particular database relation. Variables can take values of individual tuples from the relation.
A simple relational calculus query is in the form-

{t | condition (t)}

t – tuple variable

condition (t) – is a conditional expression involving *t*.

Result is a set of all tuples that satisfy the conditions specified in *condition (t)*.

Ex. Find all employees whose salary is above 50,000

$$\{t \mid EMP(t) \text{ and } t.\text{salary} > 50{,}000\}$$

Selects all tuples from EMP such that for each tuple selected, the salary value is > 50,000.

The expression EMP(*t*) specifies from where the tuple *t* must be chosen.

Hence EMP relation in this case is known as a *range relation*.

Note: The above query retrieves all the attributes of relation EMP.

The *universal* (∀), and *existential* (∃) quantifiers can be applied to tuples.

Ex.:

{t.name, t age | EMP (t) and (∃d) (Dept (d)and d.dname = 'Research' and d.dno = t.dno)}

To retrieve the name and age of all employees who work for 'Research' department.

If the tuple variable *t* occurs with ∃ or ∀ quantifiers the variable is known as *bound variable and* otherwise called as *free variable.*

Relational Completeness:

This notion is used to compare high level query languages.

Any relational query language L is considered to be *relationally complete* if we can express in L any query that is expressed in relational algebra (RA) or relational calculus (RC).

Summary

- ✓ *What is a query language*
- ✓ *Formal query languages for Relational data model*
- ✓ *Basic concepts of Relational Algebra*
- ✓ *Operations in Relational Algebra*
- ✓ *Relational Calculus*
- ✓ *Examples*