Abstract Query Languages:

Relational Algebra and Relational Calculus

Dr. Sambit Bakshi

NIT Rourkela

June 7, 2021

Outline

Introduction to Query Languages

- The schema of a relation defines its structure
- The relationships define constraints on the attributes / tuples

What would we do with the schema and constraints if there is no way to manage data in a relation!

Query languages help in manipulate data in a relation

Introduction to Query Languages

Database Model	Query Model
Relational Model	Relational Algebra (Procedural)
	and
	Relational Calculus (Non-procedural / Declarative)

These Query models are **abstract** or **theoretical**.

Relational Algebra and Relational Calculus (with only safe queries) are equivalent.

In practice, Relational model is implemented as RDBMS, and the query model is implemented as SQL.

A language is called **relationally complete** if it has capacity to express every query expressible through relational algebra. SQL is **relationally complete**.

Introduction to Query Languages

Cardinality and Degree / Arity of a Relation

- **Degree / Arity of a relation** is the number of attributes / columns of the relation.
  It is an intrinsic property of a relation.
- **Cardinality of a relation** is the number of tuples / rows of the relation.
  It is not an intrinsic property of a relation, and represents a state of the relation.

Basic Operations

There are SIX basic operations:

Unary operations:

- **Selection** (σ): Selects a subset of tuples / rows from a relation
- **Projection** (π): Retains a subset of columns from a relation
- **Rename** (ρ): Renames a relation

Binary operations:

- **Cross Product / Cartesian Product** ($\times$): Allows to combine two relations
- **Difference / Set difference** ($-$): Returns tuples present in one relation but not in the other
- **Union** ($\cup$): Consolidates tuples in two relations

More Operations

- **Intersection** ($\cap$): Returns tuples present in both relations
- **Join** ($\bowtie$): Allows to combine two or more relations
  There are several types of joins: semi join ($\rtimes, \ltimes$), left outer join (⟕), right outer join (⟖), full outer join (⟗)
- **Division** ($/$): Returns those tuples of a relation which subsets tuples of another table
- **Assignment** ($\leftarrow$): Assigning the output to a relation

Selection operation

Selection (σ): selects a subset of tuples / rows from a relation.
Also called as **horizontal partitioning**.
Syntax: $\sigma_{\texttt{<CONDITION>}}(\texttt{<RELATION>})$

EMP(EID, ENAME, ESAL, DEPT)

EID	ENAME	ESAL	DEPT
001	ABC	10500	2
002	ABD	9000	4
003	DEF	11000	4
004	DBC	12000	4

Select details of employees with salary (ESAL) greater than INR 10000 from the table EMP(EID, ENAME, ESAL, DEPT) given beside

$\sigma_{\texttt{ESAL > 10000}}(\texttt{EMP})$

EID	ENAME	ESAL	DEPT
001	ABC	10500	2
003	DEF	11000	4
004	DBC	12000	4

Select details of employees working in department (DEPT) '4' from the table EMP mentioned beside

$\sigma_{\texttt{DEPT = 4}}(\texttt{EMP})$

EID	ENAME	ESAL	DEPT
002	ABD	9000	4
003	DEF	11000	4
004	DBC	12000	4

Selection operation

Selection (σ): selects a subset of tuples / rows from a relation.
Also called as **horizontal partitioning**.
Syntax: $\sigma_{\texttt{<CONDITION>}}(\texttt{<RELATION>})$

EMP(EID, ENAME, ESAL, DEPT)

EID	ENAME	ESAL	DEPT
001	ABC	10500	2
002	ABD	9000	4
003	DEF	11000	4
004	DBC	12000	4

Select details of employees of department (DEPT) '4' having salary (ESAL) greater than INR 10000 from the table EMP mentioned beside

$\sigma_{\texttt{DEPT = 4 AND ESAL > 10000}}(\texttt{EMP})$ (cascading conditions)
$\sigma_{\texttt{DEPT = 4}}(\sigma_{\texttt{ESAL > 10000}}(\texttt{EMP}))$ (nested conditions)
$\sigma_{\texttt{ESAL > 10000}}(\sigma_{\texttt{DEPT = 4}}(\texttt{EMP}))$ (nested conditions)

EID	ENAME	ESAL	DEPT
003	DEF	11000	4
004	DBC	12000	4

As order of nesting of conditions does not affect the output of Select (σ) operator, it is **commutative**

Selection operation

- Selection operator works on **a single tuple at a time**
- The CONDITION involved in selection operation MUST be a boolean expression (or a logical combination of more than one boolean expressions)
- Selection operator may reduce the cardinality of a relation when it acts on, i.e., $|\sigma_{\texttt{<CONDITION>}}(\texttt{<RELATION>})| \leq |\texttt{<RELATION>}|$
- Selection operator does not alter the arity of a relation
- Selection operator can act on a relation or any relational algebra expression resulting on a table
- Selection operator is commutative

Projection operation

Projection (π): selects a subset of attributes from a relation.
Also called as **vertical partitioning**.
Syntax: $\pi_{\texttt{<ATTRIBUTES>}}(\texttt{<RELATION>})$

EMP(EID, ENAME, ESAL, DEPT)

EID	ENAME	ESAL	DEPT
001	ABC	10500	2
002	ABD	9000	4
003	DEF	11000	4
004	DBC	12000	4

Extract Employee IDs EID and Names ENAME of employees from the table EMP(EID, ENAME, ESAL, DEPT) given beside

$\pi_{\texttt{EID,ENAME}}(\texttt{EMP})$

EID	ENAME
001	ABC
002	ABD
003	DEF
004	DBC

Extract departments DEPT of employees from the table EMP mentioned above.

$\pi_{\texttt{DEPT}}(\texttt{EMP})$

DEPT
2
4

Removes duplicates!

Projection operation

Is Projection (π) operation commutative?

We know, for a projection operation to work, the attributes it retrieves must be a subset of the attributes of the input relation.
For $\pi_{\texttt{<ATTRIBUTES}_2\texttt{>}}(\pi_{\texttt{<ATTRIBUTES}_1\texttt{>}}(\texttt{<RELATION>}))$ to be valid, the following condition must be true:
$\texttt{<ATTRIBUTES}_2\texttt{>} \subseteq \texttt{<ATTRIBUTES}_1\texttt{>}$
Also, for projection to be commutative,
$\pi_{\texttt{<ATTRIBUTES}_2\texttt{>}}(\pi_{\texttt{<ATTRIBUTES}_1\texttt{>}}(\texttt{<RELATION>})) = \pi_{\texttt{<ATTRIBUTES}_1\texttt{>}}(\pi_{\texttt{<ATTRIBUTES}_2\texttt{>}}(\texttt{<RELATION>}))$
However, $\pi_{\texttt{<ATTRIBUTES}_2\texttt{>}}(\pi_{\texttt{<ATTRIBUTES}_1\texttt{>}}(\texttt{<RELATION>})) = \pi_{\texttt{<ATTRIBUTES}_2\texttt{>}}(\texttt{<RELATION>})$
Hence projection is not commutative.

Projection operation

- Projection operator can act on a relation or any relational algebra expression resulting on a table
- The `<ATTRIBUTES>` involved in projection operation MUST be a subset (may be equal) of all attributes in `<RELATION>`
- Projection operator may reduce the arity / degree of a relation when it acts on,
  i.e., Arity of $\pi_{\texttt{<ATTRIBUTES>}}(\texttt{<RELATION>}) \leq$ Arity of `<RELATION>`
- Projection operator may alter the cardinality of a relation as it removes duplicate tuples from output
  i.e., $|\pi_{\texttt{<ATTRIBUTES>}}(\texttt{<RELATION>})| \leq |\texttt{<RELATION>}|$
  If `<ATTRIBUTES>` is a superkey, then
  $|\pi_{\texttt{<ATTRIBUTES>}}(\texttt{<RELATION>})| = |\texttt{<RELATION>}|$
  Projection operation defined in abstract relational algebra language eliminates duplicates, but projection operator in SQL retains duplicates!
  Projection operation is not commutative.

Cascading Selection and Projection operation

Extract Employee IDs EID and Names ENAME of employees whose salary ESAL is greater than INR 10000 from the table EMP(EID, ENAME, ESAL, DEPT) given beside

EMP(EID, ENAME, ESAL, DEPT)

EID	ENAME	ESAL	DEPT
001	ABC	10500	2
002	ABD	9000	4
003	DEF	11000	4
004	DBC	12000	4

$\mathtt{TEMP} \leftarrow \sigma_{\mathtt{ESAL}\ >\ 10000}(\mathtt{EMP})$
$\mathtt{ANS} \leftarrow \pi_{\mathtt{EID,ENAME}}(\mathtt{TEMP})$

OR IT CAN ALSO BE WRITTEN INLINE AS:
$\pi_{\mathtt{EID,ENAME}}(\sigma_{\mathtt{ESAL}\ >\ 10000}(\mathtt{EMP}))$

EID	ENAME
001	ABC
003	DEF
004	DBC

- Please note that $\pi_{\mathtt{EID,ENAME}}(\sigma_{\mathtt{ESAL}\ >\ 10000}(\mathtt{EMP})) \neq \sigma_{\mathtt{ESAL}\ >\ 10000}(\pi_{\mathtt{EID,ENAME}}(\mathtt{EMP}))$, i.e., the roles of selection and projection cannot be commutative. Always selection acts first, and then projection.

Rename operation

- Syntax: Renaming relation name and attributes:
  $\rho_{\texttt{<OUTPUT\_RELATION>(<OUTPUT\_ATTRIBUTES>)}}(\texttt{<INPUT\_RELATION>})$
- Syntax: Renaming relation name only (and not attributes):
  $\rho_{\texttt{<OUTPUT\_RELATION>}}(\texttt{<INPUT\_RELATION>})$
- Syntax: Renaming attributes only (and not relation name):
  $\rho_{\texttt{(<OUTPUT\_ATTRIBUTES>)}}(\texttt{<INPUT\_RELATION>})$
- The input for rename operator can be a relation or a relational algebra expression
- Rename operation is a special type of projection

Set operations

The set operations Union ($\cup$), Intersection ($\cap$), Difference ($-$) are binary Operations.

Union ($\cup$): Consolidates tuples in two relations

Intersection ($\cap$): Extracts only common tuples from two relations

Difference ($-$): Extracts tuples present exclusively in first relation (but not present in second relation)

Set operations

Union Compatibility: The two input relations should have (i) same degree and (ii) same domain for every attribute

Union Compatibility is decided based on intrinsic properties of two relations, not based on it's state.

$\mathtt{R} \cup \mathtt{S}$ automatically stores the result in first operand, i.e., $\mathtt{R}$.

Union eliminates duplicates from the output and returns the common tuples only once in output, and intersection cannot have duplicates.

Intersection is a derived operation, i.e.
$\mathtt{R} \cap \mathtt{S} = \mathtt{R} - (\mathtt{R} - \mathtt{S}) = \mathtt{S} - (\mathtt{S} - \mathtt{R})$

Set operations

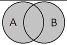
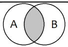
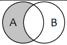
Union and intersection operations are commutative, i.e.
$\mathtt{R} \cup \mathtt{S} = \mathtt{S} \cup \mathtt{R}$ and $\mathtt{R} \cap \mathtt{S} = \mathtt{S} \cap \mathtt{R}$

Union and intersection operations are associative, i.e.
$(\mathtt{R} \cup \mathtt{S}) \cup \mathtt{T} = \mathtt{R} \cup (\mathtt{S} \cup \mathtt{T})$ and $(\mathtt{R} \cap \mathtt{S}) \cap \mathtt{T} = \mathtt{R} \cap (\mathtt{S} \cap \mathtt{T})$

Difference / minus is not commutative, i.e.,
$\mathtt{R} - \mathtt{S} \neq \mathtt{S} - \mathtt{R}$

Cardinality analysis of set operations

Inputs: `A` having cardinality `m`, `B` having cardinality `n`

Operation ↓	**Venn Diagram Representation** ↓	**Minimum Cardinality**	**Maximum Cardinality**
$A \cup B$		`max(m,n)`	`m+n`
$A \cap B$		0	`min(m,n)`
$A - B$		0	`m`

Cross Product / Cartesian Product

Cross product / Cartesian product ($\times$) is a binary Operation.

The input relations need not be union compatible for participating in cross product.

Arity of $\mathtt{R} \times \mathtt{S}$ = (Arity of $\mathtt{R}$ + Arity of $\mathtt{S}$)

Cardinality of $\mathtt{R} \times \mathtt{S}$ = (Cardinality of $\mathtt{R}$ $\times$ Cardinality of $\mathtt{S}$)

Cross Product / Cartesian Product

R(A, B, C)

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

S(D, E)

D	E
d_1	e_1
d_2	e_2

$P(A, B, C, D, E) \leftarrow R(A, B, C) \times S(D, E)$

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_3	b_3	c_3	d_1	e_1
a_3	b_3	c_3	d_2	e_2

- Cross product is commutative, i.e, $R \times S = S \times R$
- Cross product is associative, i.e., $(R \times S) \times T = R \times (S \times T)$
- In most practical cases, such unconditional linking of every tuple of a relation with every tuple of another relation is meaningless.

Join

Syntax: $\texttt{P} \leftarrow \texttt{R} \bowtie_{\texttt{<join\_condition>}} \texttt{S}$

R(A, B, C)

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

S(D, E)

D	E
a_1	b_1
a_2	b_2

$\texttt{P(A, B, C, D, E)} \leftarrow \texttt{R(A, B, C)} \bowtie_{\texttt{A = D}} \texttt{S(D, E)}$

A	B	C	D	E
a_1	b_1	c_1	a_1	b_1
a_2	b_2	c_2	a_2	b_2

- Join product is commutative, i.e, $\texttt{R} \bowtie_{\texttt{<join\_condition>}} \texttt{S} = \texttt{S} \bowtie_{\texttt{<join\_condition>}} \texttt{R}$
- Arity of $\texttt{R} \bowtie \texttt{S}$ = (Arity of R + Arity of S)
- Minimum Cardinality of $\texttt{R} \bowtie \texttt{S} = 0$
- Maximum Cardinality of $\texttt{R} \bowtie \texttt{S}$ = (Cardinality of R $\times$ Cardinality of S)

Join

<join_condition> is always a boolean condition.

<join_condition> is generally not a condition where check on a attributes on a single participating relation is checked. Such conditions can always be applied before join. Why would we apply such conditions while join!

<join_condition> is usually a condition where check between attributes of both participating relations is performed.

<join_condition> is hence denoted as $\mathtt{R}_i \ \theta \ \mathtt{S}_j$ where $\mathtt{R}_i$ and $\mathtt{S}_j$ are two attributes from the relations respectively, and θ is the binary boolean operation between them. θ can be checking if they are equal, or unequal, or one is greater than other etc.

As the operation between attributes from different relations is classically represented by θ, it is also called **theta join**.

Join

<join_condition> is very common in some practical cases, as we may check relation bewteen two related attributes.

For this, a special type of join operation, called **natural join** ($*$) is used where <join_condition> is not explicitly written. An equality check between the attribute pairs having same name is done, and accordingly the result of natural join is computed.

R(A, B, C)

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

S(A)

A
a_1
a_2

P(A, B, C) $\leftarrow$ R(A, B, C) $*$ S(A)

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

Join

R(A, B, C)

A	B	C
a_1	b_1	c_1
a_1	b_2	c_1
a_2	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

S(A, B, D)

A	B	D
a_1	b_1	d_1
a_2	b_2	d_2

$P(A, B, C, D) \leftarrow R(A, B, C) * S(A, B, D)$

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2

Common attributes are not repeated in resultant relation.

If no common attribute is there for natural join, it will degrade to cross product.

PLEASE NOTE THAT IT WON'T RETURN ZERO TUPLES IN THAT CASE.

Division

Syntax: `P ← R / S`

`P ← R / S` will retain those attributes which are present in `R` but not in `S` only for those tuples where ALL values of common attributes in `R` matches any tuple of `S`

`R(A, B)`

A	B
a_1	b_1
a_2	b_1
a_1	b_2

`S(A)`

A
a_1
a_2

`P(B) ← R(A, B) / S(A)`

B
b_1

Division

R(A, B)

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
a_1
a_2
a_3

$P(B) \leftarrow R(A, B) \; / \; S(A)$

B
b_1
b_4

$P \times S$ may not return R.

Division is not commutative, i.e., $R \; / \; S \neq S \; / \; R$.

Division

R(A, B, C)

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_3	b_1	c_3
a_4	b_1	c_2
a_1	b_2	c_1
a_3	b_2	c_1
a_2	b_3	c_1
a_3	b_3	c_1
a_4	b_3	c_1
a_1	b_4	c_2
a_2	b_4	c_2
a_3	b_4	c_3

S(A, C)

A	C
a_1	c_1
a_2	c_1
a_3	c_3

P(B) $\leftarrow$ R(A, B, C) / S(A, C)

B
b_1

Is division associative? Draw a Venn Diagram of set of three relations and check if R/(S/T) and (R/S)/T contains same set of attributes.

Division

Division is not a basic operation, that means, division operation should be expressed in terms of basic operations.

To express division ($\mathtt{T} \leftarrow \mathtt{R} \ / \ \mathtt{S}$) with basic operations:

Step# 1: $\mathtt{T1} \leftarrow \pi_{\mathtt{(R-S)}} \ \mathtt{(R)}$

Step# 2: $\mathtt{T2} \leftarrow \pi_{\mathtt{(R-S)}} \ \mathtt{((S} \times \mathtt{T1)} \ \mathtt{-} \ \mathtt{R)}$

Step# 3: $\mathtt{T} \leftarrow \mathtt{T1} \ \mathtt{-} \ \mathtt{T2}$

Outer Join

All join functions learn before are **inner join**, in which the dangling tuples (those tuples of one relation not having a referred value in other relation) does not appear in output.

Sometimes we may need those tuples which does not have a match in other relation.

A join providing such tuples in output is called **outer join**.

left outer join ($R ⟕ S$) retains the dangling tuples of R but not those of S

right outer join ($R ⟖ S$) retains the dangling tuples of S but not those of R

full outer join ($R ⟗ S$) retains the dangling tuples of both R and S

Outer join

R(A, C)

A	C
1	3
2	4

S(B, D)

B	D
1	6
5	7

R $⟕_{A=B}$ S

A	C	B	D
1	3	1	6
2	4	NULL	NULL

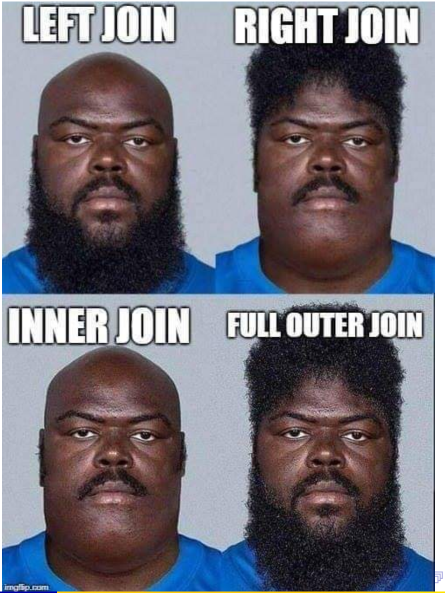
R $⟖_{A=B}$ S

A	C	B	D
1	3	1	6
NULL	NULL	5	7

R $⟗_{A=B}$ S

A	C	B	D
1	3	1	6
2	4	NULL	NULL
NULL	NULL	5	7

A meme I found on internet!



Semi-join

A **semi-join** between two relations returns tuples from the one relation where one or more matches (according to join condition) are found in tuples of the other relation.

Though a tuple in one relation may have match with more than one tuples of the other relation, the tuple will appear only once in output.

left semi-join ($R \ltimes S$) retains those tuples of R which have a match with S

right semi-join ($R \rtimes S$) retains those tuples of S which have a match with R

semi-join is not commutative.

Semi-join

R(A, B, C)

A	B	C
a_1	b_1	c_1
a_1	b_2	c_1
a_2	b_1	c_1
a_3	b_2	c_2
a_4	b_3	c_3

S(D, E, F)

D	E	F
a_1	e_1	f_1
a_1	e_2	f_3
a_2	e_2	f_2
a_5	e_2	f_3

$P(A, B, C) \leftarrow R(A, B, C) \ltimes_{A=D} S(D, E, F)$

A	B	C
a_1	b_1	c_1
a_1	b_2	c_1
a_2	b_1	c_1

$Q(D, E, F) \leftarrow R(A, B, C) \rtimes_{A=D} S(D, E, F)$

D	E	F
a_1	e_1	f_1
a_1	e_2	f_3
a_2	e_2	f_2

Anti-join

A **anti-join** or **anti semi-join** between two relations returns those tuples from the one relation for which there exists no corresponding matches (according to join condition) in tuples of the other relation.

left anti-join or called ONLY anti-join ($R \triangleright S$) retains those tuples of R which do not have a match with S

$R \triangleright S = R - (R \ltimes S)$

right anti-join ($R \triangleleft S$) retains those tuples of S which do not have a match with R

$R \triangleleft S = S - (R \rtimes S)$

anti-join is not commutative.

Anti-join

R(A, B, C)

A	B	C
a_1	b_1	c_1
a_1	b_2	c_1
a_2	b_1	c_1
a_3	b_2	c_2
a_4	b_3	c_3

S(D, E, F)

D	E	F
a_1	e_1	f_1
a_1	e_2	f_3
a_2	e_2	f_2
a_5	e_2	f_3

$P(A, B, C) \leftarrow R(A, B, C) \triangleright_{A=D} S(D, E, F)$

A	B	C
a_3	b_2	c_2
a_4	b_3	c_3

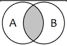
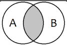
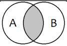
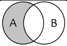
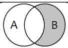
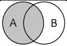
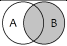
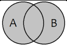
$Q(D, E, F) \leftarrow R(A, B, C) \triangleleft_{A=D} S(D, E, F)$

D	E	F
a_5	e_2	f_3

Cardinality analysis of join operations

Inputs: A having cardinality m, B having cardinality n

Note: The table bears representative Venn diagram, it does not state which attributes will appear on result, and whether tuples will be duplicated on result.

Operation ↓	Venn Diagram Representation ↓	Minimum Cardinality	Maximum Cardinality
$A \times B$	NO CONDITION	mn	mn
$A \bowtie B$		0	mn
$A \ltimes B$		0	m
$A \rtimes B$		0	n
$A \triangleright B$		0	m
$A \triangleleft B$		0	n
A ⟕ B		m	mn
A ⟖ B		n	mn
A ⟗ B		max(m,n)	mn

Problems on Relational Algebra

Q. Which of the following query transformations (i.e. replacing the l.h.s. expression by the r.h.s. expression) are incorrect?
$\mathtt{R}_1$ and $\mathtt{R}_2$ are relations, $\mathtt{C}_1$ and $\mathtt{C}_2$ are selection conditions, and $\mathtt{A}_1$ and $\mathtt{A}_2$ are attributes of $\mathtt{R}_1$.
(a) $\sigma_{\mathtt{C}_1}(\sigma_{\mathtt{C}_2}(\mathtt{R}_1)) \rightarrow \sigma_{\mathtt{C}_2}(\sigma_{\mathtt{C}_2}(\mathtt{R}_1))$
(b) $\sigma_{\mathtt{C}_1}(\pi_{\mathtt{A}_1}(\mathtt{R}_1)) \rightarrow \pi_{\mathtt{A}_1}(\sigma_{\mathtt{C}_1}(\mathtt{R}_1))$
(c) $\sigma_{\mathtt{C}_1}(\mathtt{R}_1 \cup \mathtt{R}_2) \rightarrow \sigma_{\mathtt{C}_1}(\mathtt{R}_1) \cup \sigma_{\mathtt{C}_1}(\mathtt{R}_2)$
(d) $\pi_{\mathtt{A}_2}(\sigma_{\mathtt{C}_1}(\mathtt{R}_1)) \rightarrow \sigma_{\mathtt{C}_1}(\pi_{\mathtt{A}_2}(\mathtt{R}_1))$
[GATE1998]

Problems on Relational Algebra

Q. Suppose $\mathtt{R_1(A,B)}$ and $\mathtt{R_2(C,D)}$ are two relation schemas. Let $\mathtt{r_1}$ and $\mathtt{r_2}$ be the corresponding relation instances. $\mathtt{B}$ is a foreign key that refers to $\mathtt{C}$ in $\mathtt{R_2}$. If the data in $\mathtt{r_1}$ and $\mathtt{r_2}$ satisfy referential integrity constraints, which of the following is ALWAYS TRUE?
(a) $\pi_{\mathtt{B}}(\mathtt{r_1}) - \pi_{\mathtt{C}}(\mathtt{r_2}) = \phi$
(b) $\pi_{\mathtt{C}}(\mathtt{r_2}) - \pi_{\mathtt{B}}(\mathtt{r_1}) = \phi$
(c) $\pi_{\mathtt{B}}(\mathtt{r_1}) = \pi_{\mathtt{C}}(\mathtt{r_2})$
(d) $\pi_{\mathtt{B}}(\mathtt{r_1}) - \pi_{\mathtt{C}}(\mathtt{r_2}) \neq \phi$
[GATE2012]

Problems on Relational Algebra

Q. Give a relational algebra expression using only the minimum number of operators from $\{\cup\ ,\ -\ \}$ which is equivalent to $\mathtt{R} \cap \mathtt{S}$.
[GATE1994]

Problems on Relational Algebra

Q. Consider a join of a relation `R` with a relation `S`. If `R` has `m` tuples and `S` has `n` tuples, then the maximum and minimum sizes of the join respectively are:
(a) `m+n` and 0
(b) `mn` and 0
(c) `m+n` and `|m-n|`
(d) `mn` and `m+n`
[GATE1999]

Q. Consider the following relation P(X, Y, Z), Q(X, Y, T) and R(Y, V):
How many tuples will be returned by the following relational algebra query?
$\pi_{\mathtt{X}}(\sigma(_{\mathtt{P.Y=R.Y} \wedge \mathtt{R.V=V}}(\mathtt{P} \times \mathtt{R}))) - \pi_{\mathtt{X}}(\sigma(_{\mathtt{Q.Y=R.Y} \wedge \mathtt{Q.T}>2}(\mathtt{Q} \times \mathtt{R})))$

(a) 3
(b) 1
(c) 2
(d) 4
[GATE2019]
Ans: Option (b)

P		
X	Y	Z
X1	Y1	Z1
X1	Y1	Z2
X2	Y2	Z2
X2	Y4	Z4

Q		
X	Y	T
X2	Y1	2
X1	Y2	5
X1	Y1	6
X3	Y3	1

R	
Y	V
Y1	V1
Y3	V2
Y2	V3
Y2	V2

Types of Relational Calculus

(a) Tuple Relational Calculus (TRC)
(b) Domain Relational Calculus (DRC)

TRC and DRC are **equivalent** in power (when only safe queries are concerned), i.e., any query expressed through TRC do have a parallel query in DRC that serve the same purpose.

TRC and DRC are also equivalent in power with relational algebra.

Tuple Relational Calculus

Syntax: {`t.attributes` | range relation of `t` AND conditions(`t`)}

P(A,B,C)

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_3	b_1	c_3
a_4	b_1	c_2
a_1	b_2	c_1
a_3	b_2	c_2
a_2	b_3	c_1
a_3	b_3	c_1
a_4	b_3	c_1
a_1	b_4	c_2

Find those values of `A`s for which `C` has a value c_1

{`t.A` | `P(t)` AND `t.C = ` c_1}

A
a_1
a_2
a_3
a_4

Tuple Relational Calculus

Atomic expressions:
Range relation:
$\mathtt{R(t)}$
Conditions:
$\mathtt{t.A}\ \theta\ \mathtt{constant}$
$\mathtt{t_1.A}\ \theta\ \mathtt{t_2.B}$

Composite expressions:
Range relation:
$\mathtt{R_1(t_1)} \wedge \mathtt{R_2(t_2)} \wedge \ldots \wedge \mathtt{R_n(t_n)}$
Conditions:
(i) with free variables:
$\mathtt{F_1} \wedge \mathtt{F_2}$, $\mathtt{F_1} \vee \mathtt{F_2}$, $\neg\ \mathtt{F_1}$
(ii) with bounded variables using quantifiers: $\forall\ \mathtt{t(F)}$, $\exists\ \mathtt{t(F)}$

Tuple Relational Calculus

List the names and addresses of all employees who work for department named `'CSE'`. The database schema is as following:

EMP(<u>ENAME</u>, EADDRESS, EDOB, ESAL, DNO)

DEPT(<u>DNO</u>, DNAME, DMGR)

EMP.DNO is foreign key to DEPT.DNO

$$\{\texttt{t.ENAME,t.EADDRESS} \mid \texttt{EMP(t)} \wedge (\exists\ \texttt{d})(\texttt{DEPT(d)} \wedge \texttt{d.DNAME='CSE'} \wedge \texttt{d.DNO=t.DNO})\}$$

Domain Relational Calculus

Syntax: {<attributes> | domain constraint of attributes AND conditions on attributes}

P(A,B,C)

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_3	b_1	c_3
a_4	b_1	c_2
a_1	b_2	c_1
a_3	b_2	c_2
a_2	b_3	c_1
a_3	b_3	c_1
a_4	b_3	c_1
a_1	b_4	c_2

Find those values of As for which C has a value c_1

$\{<a> \mid \exists b\ \exists c\ (P(a,b,c) \text{ AND } c = c_1)\}$

A
a_1
a_2
a_3
a_4

Domain Relational Calculus

Formal Definition: An expression in the domain relational calculus is of the form
$\{<\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n> \mid P(\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n)\}$
where $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$ represent domain variables, P represents a formula composed of atoms, as was the case in the tuple relational calculus.
An atom in the domain relational calculus has one of the following forms:

- $<\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n> \in r$, where r is a relation on n attributes and $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$ are domain variables or domain constants.
- $\mathtt{x}\ \theta\ \mathtt{y}$, where $\mathtt{x}$ and $\mathtt{y}$ are domain variables and θ is a comparison operator. We require that attributes x and y have domains that can be compared by θ.
- $\mathtt{x}\ \theta\ \mathtt{c}$, where $\mathtt{x}$ is a domain variable, θ is a comparison operator, and $\mathtt{c}$ is a constant in the domain of the attribute for which $\mathtt{x}$ is a domain variable.

We build up formulae from atoms by using the following recursively applied rules:

- An atom is a formula.
- If P is a formula, then so are $\neg P$ and (P).
- If $P1$ and $P2$ are formulae, then so are $P1 \wedge P2$, $P1 \vee P2$, and $P1 \Rightarrow P2$
- If $P(\mathtt{x})$ is a formula in $\mathtt{x}$, where $\mathtt{x}$ is a domain variable, then: $\exists \mathtt{x}(P(\mathtt{x}))$ and $\forall \mathtt{x}(P(\mathtt{x}))$ are also formulae.