Database Engineering

Lecture #5

Query Languages: Relational Algebra

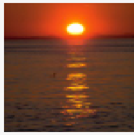
Presented By:

Dr. Suvasini Panigrahi

Associate Professor, Department of CSE,

VSSUT, Burla

Database System Concepts, 5th Ed.



Query Languages

- Fundamental Relational-Algebra-Operations
- Additional Relational-Algebra-Operations
- Extended Relational-Algebra-Operations

Query Languages

- Language in which user requests information from the database.
- Categories of query languages are as follows:
  - Procedural Language
  - Non-Procedural / Declarative Language
  - Procedural Language
  - Relational Algebra
  - Non-Procedural / Declarative Language
  - Tuple Relational Calculus (TRC)
  - Domain Relational Calculus (DRC)

Fundamental Relational Algebra Operations

- Procedural language
- The six fundamental relational algebra operations include:
  - **Select: σ**
  - **Project: ∏**
  - **Union: ∪**
  - **Set difference: –**
  - **Cartesian product: x**
  - **Rename: *ρ***
- The operators take one or two relations as inputs and produce a new relation as a result.

Select Operation

- Notation: $\sigma_{p}(r)$
- p is called the **selection predicate**
- Defined as:

$$\sigma_p(\mathbf{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

Where p is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)

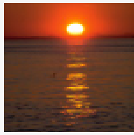
- Each **comparison/term** is of the form:

<attribute> *op* <attribute> or <constant>

where *op* is one of: =, ≠, >, ≥. <. ≤

- Example of selection:

$$\sigma_{branch\_name=\text{“Delhi”}}(account)$$



Select Operation – Example

- Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A=B \wedge D > 5}\,(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Project Operation

- Notation:

$$\prod_{A1,\ A2,\ \ldots,\ Ak} (r)$$

where $A_1, A_2, \ldots, A_k$ are attribute names and r is a relation name.

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows are removed from result, since relations are sets
- Example: To eliminate the *branch_name* attribute of *account*

$$\prod_{account\_number,\ balance} (account)$$

Project Operation – Example

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\prod_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Union Operation

- Notation: $r \cup s$
- Defined as:

  $$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.
  1. *r, s* must have the *same* **arity** (same number of attributes)
  2. The attribute domains must be **compatible** (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)
- Example: to find all customers with either an account or a loan

  $$\Pi_{customer\_name}\,(depositor) \cup \Pi_{customer\_name}\,(borrower)$$

Union Operation – Example

- Relations *r, s:*

A	*B*
α	1
α	2
β	1

r

A	*B*
α	2
β	3

s

- r ∪ s:

A	*B*
α	1
α	2
β	1
β	3

Set Difference Operation

- Notation $r - s$
- Defined as:

$$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

- Set differences must be taken between **compatible** relations.
  - r and s must have the same arity
  - attribute domains of r and s must be compatible

Set Difference Operation – Example

- Relations *r*, *s*:

A	*B*
α	1
α	2
β	1

r

A	*B*
α	2
β	3

s

- *r* – *s*:

A	*B*
α	1
β	1

Cartesian-Product Operation

- The Cartesian-product operation associates *every tuple of one relation r with every tuple of the other relation s.*
- Notation: ***r* x *s***. The Cartesian product operation is defined as:

  $$r \times s = \{t\,q \mid t \in r \text{ and } q \in s\}$$

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \emptyset$).
- If attributes of *r(R)* and *s(S)* are not disjoint, then a naming scheme must be used to distinguish between the attributes with same names. This is done by attaching the name of the relation to an attribute from which the attribute originally came.
- In general, if we have relations *r1(R1) and r2(R2), then r = r1 × r2 is a relation whose* schema is the concatenation of *R1 and R2. The result relation contains all combinations of tuples for which* there is a tuple *t1 in r1 and a tuple t2 in r2.*
- Assume that we have *n*1 tuples in *r1* and *n*2 tuples in *r*2. Then, there are *n*1 ⋆ *n*2 ways of choosing a pair of tuples — one tuple from each relation; so there are ***n*1 ⋆ *n*2 tuples in *r*.**

Cartesian-Product Operation – Example

- Relations *r, s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Rename Operation

- Rename operation allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- It also allows us to refer to a relation by more than one name.
- Notation:

$$\rho_X(E)$$

returns the relational-algebra expression *E* under the name *X.*

- If a relational-algebra expression *E* has arity *n*, then

$$\rho_{x(A_1, A_2, ..., A_n)}(E)$$

returns the result of expression *E* under the name *X*, and with the attributes renamed to A_1, A_2, …., A_n .

Banking Example

branch (branch_name, branch_city, assets)

customer (customer_name, customer_street, customer_city)

account (account_number, branch_name, balance)

loan (loan_number, branch_name, amount)

depositor (customer_name, account_number)

borrower (customer_name, loan_number)

Example Queries

- **Find all loans of over $1200.**

$$\sigma_{amount\ >\ 1200}\ (loan)$$

- **Find the loan number for each loan of an amount greater than $1200.**

$$\Pi_{loan\_number}\ (\sigma_{amount\ >\ 1200}\ (loan))$$

- **Find the names of all customers who have a loan, an account, or both from the bank.**

$$\Pi_{customer\_name}\ (borrower) \cup \Pi_{customer\_name}\ (depositor)$$

Example Queries

- **Find the names of all customers who have a loan at the Delhi branch.**

$$\Pi_{customer\_name}\,(\sigma_{branch\_name=\text{“Delhi”}}\,(\sigma_{borrower.loan\_number\,=\,loan.loan\_number}(borrower \times loan)))$$

- **Find the names of all customers who have a loan at the Delhi branch but do not have an account at any branch of the bank.**

$$\Pi_{customer\_name}\,(\sigma_{branch\_name\,=\,\text{“Delhi”}}\,(\sigma_{borrower.loan\_number\,=\,loan.loan\_number}(\text{borrower} \times \text{loan}))) \;-\; \Pi_{customer\_name}(\text{depositor})$$

Example Queries

- **Find the names of all customers who have a loan at the Delhi branch.**

  loan (loan_number, branch_name, amount)
  borrower (customer_name, loan_number)

  - **Query 1**

    $$\Pi_{customer\_name}\ (\sigma_{branch\_name\ =\ \text{“Delhi”}}$$
    $$(\sigma_{borrower.loan\_number\ =\ loan.loan\_number}\ (borrower \times loan)))$$

  - **Query 2**

    $$\Pi_{customer\_name}(\sigma_{loan.loan\_number\ =\ borrower.loan\_number}$$
    $$((\sigma_{branch\_name\ =\ \text{“Delhi”}}\ (loan)) \times\ borrower))$$

Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\prod_S(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_{x} (E_1)$, x is the new name for the result of E_1

Additional Relational Algebra Operations

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

- **Set Intersection**
- **Natural Join**
- **Division**
- **Assignment**

Set-Intersection Operation

- Notation: $r \cap s$
- The intersection operation is defined as:

  $$\boldsymbol{r \cap s = \{ t \mid t \in r \text{ and } t \in s \}}$$

- Assume:
  - r, s have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$

Set-Intersection Operation – Example

- Relation *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Natural-Join Operation

- A join operation pairs two tuples from two different relations, if and only if a given join condition is satisfied.
- We can perform a **Natural Join** only if there is at least one common attribute existing between the two relations.
- The common attributes must have the same name and domain in both the relations.
- Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Natural-Join Operation

- Notation: **r** ⋈ **s**
  - Let *r* and *s* be relations on schemas *R* and *S* respectively. Then, **r** ⋈ **s** is a relation on schema $R \cup S$ obtained as follows:
    - Consider each pair of tuples t_r from *r* and t_s from *s*.
    - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result.
    - Example:

      $R = (A, B, C, D)$

      $S = (E, B, D)$
    - Result schema = (A, B, C, D, E)
    - **r** ⋈ **s** is defined as:

      $$\Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}\ (\sigma_{r.B = s.B \wedge r.D = s.D}\ (r \times s))$$

Natural Join Operation – Example

- Relations r, s:

A	*B*	*C*	*D*
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	*D*	*E*
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	∈

s

- r ⋈ s

A	*B*	*C*	*D*	*E*
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Natural-Join Operation Example

Table 1: Student

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

Table 2: Student_Sports

ROLL_NO	SPORTS
1	Badminton
2	Cricket
2	Badminton
4	Badminton

Natural-Join Operation Example

- Student ⋈ Student_Sports
- In terms of basic operators, the natural join is done by taking cross product, selection and projection operations.

$$\Pi_{(STUDENT.ROLL\_NO,\ STUDENT.NAME,\ STUDENT.ADDRESS,\ STUDENT.PHONE,\ STUDENT.AGE\ STUDENT\_SPORTS.SPORTS)}\ (\sigma_{(STUDENT.ROLL\_NO\ =\ STUDENT\_SPORTS.ROLL\_NO)}\ (STUDENT \times STUDENT\_SPORTS))$$

- **RESULT RELATION:**

ROLL_NO	NAME	ADDRESS	PHONE	AGE	SPORTS
1	RAM	DELHI	9455123451	18	Badminton
2	RAMESH	GURGAON	9652431543	18	Cricket
2	RAMESH	GURGAON	9652431543	18	Badminton
4	SURESH	DELHI	9156768971	18	Badminton

Division Operation

- Notation: $r \div s$
- Suited to queries that include the phrase "**for all**".
- Let *r* and *s* be relations on schemas *R* and *S* respectively where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

  The result of $r \div s$ is a relation on schema:

  $R - S = (A_1, \ldots, A_m)$
- ***Division Operator (÷):*** Division operator A ÷ B can be applied if and only if:
  - Attributes of B is proper subset of attributes of A.
  - The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)
  - A ÷ B = tuples of A associated with all tuples of B.

Division Operation – Example

- Relations *r, s*:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
∈	6
∈	1
β	2

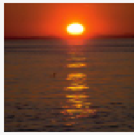
r

B
1
2

s

- $r \div s$:

A
α
β



Another Division Example

- Relations *r*, *s*:

A	*B*	*C*	*D*	*E*
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

r

D	*E*
a	1
b	1

s

- *r* ÷ *s*:

A	*B*	*C*
α	a	γ
γ	a	γ

Assignment Operation

- The assignment operation (←) provides a convenient way to express complex queries.
- The result to the right of the ← is assigned to the relation variable on the left of the assignment operator ←.
  - A complex operation can be done in the following manner:
    - A series of assignments
    - Finally, followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.
- **Example:** $r \div s$ can be done by the following steps:

$$temp1 \leftarrow \prod_{R-S} (r)$$

$$temp2 \leftarrow \prod_{R-S} ((temp1 \times s) - \prod_{R-S,S} (r))$$

$$result = temp1 - temp2$$

Banking Example

branch (branch_name, branch_city, assets)

customer (customer_name, customer_street, customer_city)

account (account_number, branch_name, balance)

loan (loan_number, branch_name, amount)

depositor (customer_name, account_number)

borrower (customer_name, loan_number)

Bank Example Queries

- Find the names of all customers who have a loan and an account at bank.

$$\Pi_{customer\_name}\ (borrower) \cap \Pi_{customer\_name}\ (depositor)$$

- Find the name of all customers who have a loan at the bank, their loan number and the loan amount.

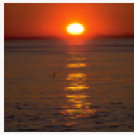
$$\Pi_{customer\_name,\ loan\_number,\ amount}\ (borrower \bowtie loan)$$

Bank Example Queries

- Find all customers who have an account from the "Delhi" and the "Kolkata" branches.
  - Query:

$$\Pi_{customer\_name}\,(\sigma_{branch\_name\,=\,\text{“Delhi”}}\,(depositor \bowtie account\,))\cap$$

$$\Pi_{customer\_name}\,(\sigma_{branch\_name\,=\,\text{“Kolkata”}}\,(depositor \bowtie account))$$



Bank Example Queries

branch (branch_name, branch_city, assets)

account (account_number, branch_name, balance)

depositor (customer_name, account_number)

Find all customers who have an account at all branches located in Brooklyn city.

- We can obtain all branches in Brooklyn by the expression:

  $$r1 = \Pi_{branch\_name}\ (\sigma_{branch\_city = \text{“Brooklyn”}}\ (branch))$$

- We can find all (*customer_name, branch_name*) pairs for which the customer has an account at a branch by writing:

  $$r2 = \Pi_{customer\_name,\ branch\_name}\ (depositor \bowtie account)$$

- Now, we need to find customers who have an account at all branches located in Brooklyn city. The operation that provides exactly those customers is the divide operation of *r2 ÷ r1* .

  $$\Pi_{customer\text{-}name,\ branch\text{-}name}\ (depositor \bowtie account) \div \Pi_{branch\text{-}name}\ (\sigma_{branch\text{-}city = \text{“Brooklyn”}}\ (branch)) = r2 \div r1$$

Extended Relational-Algebra-Operations

- Generalized Projection
- Aggregate Functions
- Outer Join

Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the attribute list.

$$\prod_{F_1, F_2, \ldots, F_n}(E)$$

- *E* is any relational-algebra expression
- Each of F_1, F_2, …, F_n are arithmetic expressions involving constants and attributes in the schema of *E*.
- Given relation *credit_info(customer_name, limit, credit_balance),* find how much more each person can spend:

$$\prod_{\textit{customer\_name, limit – credit\_balance}} (\textit{credit\_info})$$

Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- The general form of the **aggregation operation** *G* is as follows:

$$_{G_1,G_2,\ldots,G_n}\,\mathcal{G}_{\,F_1(A_1),F_2(A_2,\ldots,F_n(A_n)}(E)$$

E is any relational-algebra expression
– G_1, G_2 …, G_n is a list of attributes on which to group (can be empty)
– Each F_i is an aggregate function
– Each A_i is an attribute name

- The symbol *G* is the letter *G* in calligraphic font. The relational-algebra operation *G* signifies the aggregation function to be applied.

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

- $g_{\textbf{sum(c)}}$ (r)

sum(*c*)
27

Aggregate Operation – Example

- Relation *account* grouped by *branch-name*:

branch_name	*account_number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$_{branch\_name}\, g\, _{\mathbf{sum}(balance)}\ (account)$$

branch_name	**sum**(*balance*)
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Functions (Cont.)

- Result of aggregation does not have a name
  - We can use rename operation to give it a name in the following manner:

$$_{branch\_name}\,\mathcal{G}\ _{\mathbf{sum}(balance)\ \mathbf{as}\ sum\_balance}\,(account)$$

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons in the join condition which are **false** are inserted with *null* values.

Outer Join – Example

- Relation *loan*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

- Relation *borrower*

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

Outer Join – Example

- **Natural Join**

loan $\bowtie$ ***borrower***

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

- **Left Outer Join**

loan $⟕$ ***borrower***

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

Outer Join – Example

- **Right Outer Join**

  loan* ⟖ *borrower

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

- **Full Outer Join**

  loan* ⟗ *borrower

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations are expressed using the assignment operator.

Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only whole tuples; cannot delete values on only particular attributes
- A deletion is expressed in relational algebra by:

$$\boldsymbol{r \leftarrow r - E}$$

where r is a relation and E is a relational algebra query.

Deletion Examples

- Delete all account records in the Perryridge branch.

$$account \leftarrow account - \sigma_{branch\_name\ =\ \text{“Perryridge”}}\ (account\ )$$

- Delete all loan records with amount in the range of 0 to 50

$$loan \leftarrow loan - \sigma_{amount\ \geq\ 0\ and\ amount\ \leq\ 50}\ (loan)$$

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- in relational algebra, an insertion is expressed by:

$$\boldsymbol{r} \leftarrow \boldsymbol{r} \cup \boldsymbol{E}$$

  where r is a relation and E is a relational algebra expression.
- The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple.

Insertion Examples

- Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

$$account \leftarrow account \cup \{(\text{“A-973”}, \text{“Perryridge”}, 1200)\}$$

$$\text{depositor} \leftarrow depositor \cup \{(\text{“Smith”}, \text{“A-973”})\}$$

$\bowtie$

Updating

- A mechanism to change a value in a tuple without charging *all* values in the tuple
- Use the generalized projection operator to do this task

$$r \leftarrow \prod_{F_1, F_2, \ldots, F_l,} (r)$$

- Each F_i is either
  - The i^{th} attribute of r, if the i^{th} attribute is not updated, or,
  - If the attribute is to be updated F_i is an expression, involving only constants and the attributes of r, which gives the new value for the attribute

Update Examples

- Make interest payments by increasing all balances by 5 percent.

$$account \leftarrow \Pi_{account\_number,\ branch\_name,\ balance\ *\ 1.05}\ (account)$$

- Pay all accounts with balances over $10,000, 6 percent interest and pay all others 5 percent

$$account \leftarrow \Pi_{account\_number,\ branch\_name,\ balance\ *\ 1.06}\ (\sigma_{BAL\ >\ 10000}\ (account\ ))$$
$$\cup\ \Pi_{account\_number,\ branch\_name,\ balance\ *\ 1.05}\ (\sigma_{BAL\ \leq\ 10000}\ (account))$$