Database Management System (DBMS)

Lecture-19

Dharmendra Kumar

September 10, 2020

Generalized Projection

The generalized-projection operation extends the projection operation by allowing arithmetic functions to be used in the projection list. The generalized projection operation has the form

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

Where E is any relational-algebra expression, and each of $F_1, F_2, \ldots, F_n$ is an arithmetic expression involving constants and attributes in the schema of E. As a special case, the arithmetic expression may be simply an attribute or a constant.

Aggregate Functions

Aggregate functions take a collection of values and return a single value as a result. For example, the aggregate function **sum** takes a collection of values and returns the sum of the values. Following aggregate functions are used.

1. sum
2. avg
3. count
4. min
5. max

Relational Algebra

Example: Find the sum of all account balances.

Solution: Query for this will be

$$\mathcal{G}_{sum(balance)}(account)$$

The symbol $\mathcal{G}$ is the letter G in calligraphic font; read it as "calligraphic G." The relational-algebra operation $\mathcal{G}$ signifies that aggregation is to be applied, and its subscript specifies the aggregate operation to be applied.

Relational Algebra

Example: Find the sum of all account balances of each branch.

Solution: Query for this will be

$$_{branch-name}\mathcal{G}_{sum(balance)}(account)$$

Example: Find the number of depositors.

Solution: Query for this will be

$$\mathcal{G}_{count(customer-name)}(depositor)$$

Outer Join

The outer-join operation is an extension of the join operation to deal with missing information. There are actually three forms of the operation: left outer join, denoted ⟕; right outer join, denoted ⟖ ; and full outer join, denoted ⟗ . All three forms of outer join compute the join, and add extra tuples to the result of the join.

The left outer join (⟕) takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation, and adds them to the result of the natural join.

Relational Algebra

The right outer join ($\rtimes$) is symmetric with the left outer join: It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join.

The full outer join($\bowtie$) does both of those operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join.

Relational Algebra

Example: Consider the following two relations Employee and FT-works:-

employee-name	*street*	*city*
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
Williams	Seaview	Seattle

Employee table

employee-name	*branch-name*	*salary*
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

FT-works table

Relational Algebra

Natural join and left outer join are the followings:-

employee-name	*street*	*city*	*branch-name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500

Employee ⋈ FT-works

employee-name	*street*	*city*	*branch-name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	*null*	*null*

Employee ⟕ FT-works

Relational Algebra

Right outer join and Full outer join are the followings:-

employee-name	*street*	*city*	*branch-name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Gates	*null*	*null*	Redmond	5300

Employee ⟖ FT-works

employee-name	*street*	*city*	*branch-name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	*null*	*null*
Gates	*null*	*null*	Redmond	5300

Employee ⟗ FT-works

Modification of the Database

Deletion

We can delete only whole tuples; we cannot delete values on only particular attributes.

In relational algebra a deletion is expressed by

$$r \leftarrow r - E$$

Where r is a relation and E is a relational-algebra query.

Relational Algebra

Example: Delete all of Smith's account records.

Solution: $depositor \leftarrow depositor - \sigma_{customer-name}(depositor)$

Example: Delete all loans with amount in the range 0 to 50.

Solution: $loan \leftarrow loan - \sigma_{amount \geq 0 \ \wedge \ amount \leq 50}(loan)$

Relational Algebra

Example: Delete all accounts at branches located in Needham.

Solution:

$$account \leftarrow account - r$$

Where r is

$$r \leftarrow \Pi_{account-number, branch-name, balance}$$
$$(\sigma_{branch-city = \text{"}Needham\text{"}} (branch \bowtie account))$$

Insertion

To insert data into a relation, we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted. Obviously, the attribute values for inserted tuples must be members of the attribute's domain. Similarly, tuples inserted must be of the correct arity. The relational algebra expresses an insertion by

$r \leftarrow r \cup E$

where r is a relation and E is a relational-algebra expression.

Relational Algebra

Example: Suppose that we wish to insert the fact that Smith has $1200 in account A-973 at the Perryridge branch.

Solution:

$$depositor \leftarrow depositor \cup (\text{"}Smith\text{"}, \text{"}A-973\text{"})$$
$$account \leftarrow account \cup (\text{"}A-973\text{"}, \text{"}Perryridge\text{"}, 1200)$$