Unit -3 Relational Algebra

**Intro of Relational algebra**:

The relational algebra defines a set of operations for the relational model, the **relational calculus** provides a higher-level *declarative* language for specifying relational queries. A relational calculus expression creates a new relation. The relational algebra is often considered to be an integral part of the relational data model. Its operations can be divided into two groups. One group includes set operations from mathematical set theory; these are applicable because each relation is defined to be a set of tuples in the *formal* relational model Set operations include UNION, INTERSECTION, SET DIFFERENCE, and CARTESIAN PRODUCT (also known as CROSS PRODUCT). The other group consists of operations developed specifically for relational databases—these include SELECT, PROJECT, and JOIN, among others. First, we describe the SELECT and PROJECT operations, because they are **unary operations** that operate on single

Relations and other complex **binary operations**, which operate on two tables by combining

related tuples (records) based on *join conditions*. Some common database requests cannot be performed with the original relational algebra operations, so additional operations were created to express these requests. These include **aggregate functions**, which are operations that can *summarize* data from the tables, as well as additional types of JOIN and UNION operations, known as OUTER JOINs and OUTER UNIONs. These operations, which were added to the original relational algebra because of their importance to many database applications.

**Fundamental Operations**

The fundamental operations selection, projection and rename on one relation so they called unary operations. Others operations union, set difference and Cartesian product operates on pairs of relations and so called binary operations.

***The Selection Operation***

The Select Operation selects tuples that satisfy a given predicate. Select is denoted by a lowercase Greek letter sigma (σ ), with the predicate appearing as a subscript. The relation is specifying within parentheses afterσ . That is, general structure of selection is

σ p(r) where p is selection predicate

where p is formula in propositional calculus consisting terms connected by connectives: ^ (and), (or), ∨ ¬ (not). Each term is in the format

<attribute>op<attribute>or <constant>

where op is one of the comparison operators: =, ≠ ,<,≤,>,≥

Examples:

1. Select those tuples of loan relation where the branch is Kathmandu.

σ branch\_name=”Kathmandu”(loan)

output = {t | t[branch\_name] = Kathmandu }

2. Find all tuples in loan relation in which amount loan is more than 5000

σ amount>5000(loan)

3. Find all tuples in loan relation where amount is more than 5000 and branch is Kathmandu.

σ branch\_name=”Kathmandu” ^ amount>5000(loan)

***The projection Operation***

The projection operation retrieves tuples for specified attributes of relation. It eliminates duplicate tuples in relation. The projection is denoted by uppercase Greak letter pi (π).

We need to specify attributes that we wish to appear in the result as a subscript to Π.

The general structure of projection is

A1,A2, . . ,Ak (r)

where A1, A2, . .Ak are attributes of relation r.

Example:Find account number and their balance from account relation

π account\_number,balance(account)

output== {t | t[account\_number, balance]}

Composition of relational operations Relational algebra operations can be composed together into relational-algebra expression.

This required for complicated query.

Example: Find those customers who say in Kathmandu.

Πcustomer\_name(σ customer\_city=”Kathmandu”(customer))

***Union Operation***

Let r and s are two relations then their union defines as

r U s ={t | t∈r or t∈s}

For rU s to be valid, it must hold

**•** r,s must have same arity (same number of attributes)

• The attribute domain must be compatible (e.g. domain of Ith column of r must deals with same type of domain of ith column of s)

Example: Find all customers with either account or loan.

Πcustomer\_name(depositor) U Πcustomer\_name(borrower)

***Set difference Operation***

The set difference allows us to find tuples that are in one relation but not in another

relation. The expression r-s produces a relation containing those tuples in r but not in s.

Formally, let r and s are two relations then their difference r-s define as

r-s=={t | t∈r and t∉s}

The set difference must be taken between compatible relations. For r-s to be valid, it must

hold

• R and s must have the same arity

• Attribute domains of r and s must be compatible

Example: Find all customer of the bank who have account but not loan

π customer\_name(depositor) - π customer\_name(borrower)

Formally, let r and s are two relations then their difference r-s define as

r-s=={t | t∈r and t∉s}

The set difference must be taken between compatible relations. For r-s to be valid, it must hold

• R and s must have the same arity

• Attribute domains of r and s must be compatible

Example: Find all customer of the bank who have account but not loan

π customer\_name(depositor) - π customer\_name(borrower)

***Cartesian Product Operation***

The Cartesian product operation denoted by cross (×). It allows us to combine information from any two relations. Cartesian product of two relations r and s, denoted by r×s returns a relation instance whose schema contains all the fields of r (in same order as they appear in r( followed all field of s (in the same order as they appear in s). The result of r×s contains one tuples <r,s> (concatenation of tuples of r and s) for each pair tuples t∈r, q∈s.

Formally,

r×s={<t,q>|t∈r and q∈s}

|  |
| --- |
|  |

|  |
| --- |
|  |

Query: Find all customer who taken loan from branch “B1”.

π customer\_name(σ borrower.loan\_number=loan.loan\_number(σ branch\_name=”B1” (borrower×loan)))

Output is

|  |
| --- |
|  |

**Rename Operation :** The relations that depict operation results do not have any names. In general, for most queries, we need to apply several relational algebra operations one after the other. Either we can write the operations as a single **relational algebra expression** by nesting the operations, or we can apply one operation at a time and create intermediate result relations. In the latter case, we must give names to the relations that hold the intermediate results.

For example, to retrieve the first name, last name, and salary of all employees who work in departmentnumber 5, we must apply a SELECT and a PROJECT operation.

We can write a single relational algebra expression, also known as an **in-line expression**, as follows:

πFname, Lname, Salary(σDno=5(EMPLOYEE))

Alternatively, we can explicitly show the sequence of operations, giving a name to

each intermediate relation, as follows:

DEP5\_EMPS ← σDno=5(EMPLOYEE)

RESULT ← πFname, Lname, Salary(DEP5\_EMPS)

**Division Operation :-**

The DIVISION operation, denoted by ÷, is useful for a special kind of query that

sometimes occurs in database applications. An example is *Retrieve the names of*

*employees who work on* ***all*** *the projects that ‘John Smith’ works on*. To express this

query using the DIVISION operation, proceed as follows. First, retrieve the list of

project numbers that ‘John Smith’ works on in the intermediate relation

SMITH\_PNOS:

SMITH ← σFname=‘John’ **AND** Lname=‘Smith’(EMPLOYEE)

SMITH\_PNOS ← πPno(WORKS\_ON Essn=SsnSMITH)

Next, create a relation that includes a tuple <Pno, Essn> whenever the employee

whose Ssn is Essn works on the project whose number is Pno in the intermediate

relation SSN\_PNOS:

SSN\_PNOS ← πEssn, Pno(WORKS\_ON)

Finally, apply the DIVISION operation to the two relations, which gives the desired

employees’ Social Security numbers:

SSNS(Ssn) ← SSN\_PNOS ÷ SMITH\_PNOS

|  |
| --- |
|  |

**Additional Operations (extended operation)**

The fundamental operations of the relational algebra are sufficient to express any relational

algebra query. But for complex query it is difficult. Additional operations (set intersection,

natural join, division, assignment) simplify the common queries.

***Set-intersection operation***

Let r and s are two relation having same arity and attributes of r and s are compatible then

their intersection r ∩ s define as

r∩ s={t|t∈r and t∈s}

In terms of fundamental operation of relational algebra it can express as r∩ s=r-(r-s)

|  |
| --- |
|  |

Example 2: Find all customer who have both loan and account

customer\_name(borrower) ∩ customer\_name (depositor)

**The JOIN Operation**

The **JOIN** operation, denoted by , is used to combine *related tuples* from two relations

into single “longer” tuples. This operation is very important for any relational database with more than a single relation because it allows us to process relationships among relations.

To illustrate JOIN, suppose that we want to retrieve the name of the manager of each department. To get the manager’s name, we need to combine each department tuple with the employee tuple whose Ssn value matches the Mgr\_ssn value in the department tuple. We do this by using the JOIN operation andthen projecting the result over the necessary attributes, as follows:

DEPT\_MGR 🡨 DEPARTMENT  Mgr\_ssn=Ssn EMPLOYEE

RESULT 🡨 Dname, Lname, Fname(DEPT\_MGR)

The JOIN operation can be specified as a CARTESIAN PRODUCT operation followed

by a SELECT operation. However, JOIN is very important because it is used very frequently

when specifying database queries. Consider the earlier example illustrating

CARTESIAN PRODUCT, which included the following sequence of operations:

EMP\_DEPENDENTS ← EMPNAMES × DEPENDENT

ACTUAL\_DEPENDENTS Ssn=Essn(EMP\_DEPENDENTS)

These two operations can be replaced with a single JOIN operation as follows:

ACTUAL\_DEPENDENTS ← EMPNAMES Ssn=Essn DEPENDENT.

**Extended Relational-Algebra operations**

Generalized projection, outer join and aggregation function are extension on basic relational algebra operation.

***Generalized Projection***

Generalized projection operation allows arithmetic functions in the projection list. The general structure is F1,F2, . .Fn(E)

where E is any relational algebra expression. Each F1,F2, . .Fn are arithmetic expression

involving constraints and attributes in the schema of E.

Example 1:

Suppose a relation

credit\_info(customer\_name, credit\_limit,credit\_balance)

Query: Find how much more each person can spend.

customer\_name, credit\_limit - credit\_balance(credit\_info)

The resulting attribute from credit\_limit – credit\_balance does not have name; its name can

be specify as below

customer\_name, credit\_limit - credit\_balance as credit\_available(credit\_info).

**Aggregate function and operations**

Aggregate function takes a collection of values and return as a single value as a result.

Some aggregate functions are

• AVG: average value

• MIN: minimum value

• SUM: sum of values

• Count: number of values

. MAX : maximum value

The aggregate operation in relational algebra denoted by the symbol (i.e.  is the letter G in calligraphic font). The general structure is

G1,G2, . . ,Gn F1(A1),F2(A2), . .,Fn(An)(E)

|  |
| --- |
|  |

***Outer join***

The outer-join operation is extension to natural join. It has a capability to deal with missing

information. There are three form of outer-join operation

(a) Left outer-join ( )

Takes all tuples in the left relation. If there are any tuples in right relation that does not match with tuple in left relation, simply pad these right relation tuples with null.

Add them to the result of the left outer-join.

(b) Right outer-join( )

Takes all tuples in the right relation. If there are any tuples in the left relation that does not match with tuple in right relation, simply pad left relation tuples with null.

Add them to the result of the left outer-join.

(c) Full outer-join ( )

Pad tuples from the left relation that that did not match any from the right relation

Add them to the result of full outer-join.

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

**Data modification operation :**

Insertion, deletion and updating operations are responsible for database modification.

***Deletion***

A delete request is expressed similarity to the query, except instead of displaying tuples,

the selected tuples are removed from the databases. Delete request can delete only whole

tuples, can not delete values on only particular attributes. Deletion is expressed as

r← r - E

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

Example 1: Delete all account in the “B1” branch.

account ←account - σ branch\_name=”B1”(account)

Example 2: Delete all records with account in the range of 0 to 50.

loan←loan - σ amount>0 and amount ≤ 50(loan)

**Insert :-T**o insert data into relation we can either specify a tuple to be inserted or write a query

whose result is a set of tuples to be inserted. In relational-algebra, an insertion is express

by

r←r ∪ E

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

Example: insert information in the database specifying customer “X” has 5000 in account

A1 at the kathmandu city.

account←account ∪{(A1,”kathmandu”,5000)}

depositor←depositor∪{(“x”,A1)}

***Updating***

Updating allow to change a value in a tuple without changing all values in tuple. In

relational algebra, updating express by

r← π F1,F2, . . ,Fn(r) where each Fi is either

• ith attributes of r, if the ith attribute is not updated or

• expression involving only constant and attributes of r, if the attribute is to be

updated. It gives the new value for the attribute.

**Example 1: increase balance by 5% to all branches.**

account←Π account\_number,branch\_name,balance\*1.05(account)

**Example 2: Increase the balance by 6% for those account which balance is over 5000 and**

**for the rest of account increase balance by 5%.**

account←Π account\_number,branch\_name,balance\*1.06(σ balance>5000(account))

U Π account\_number,branch\_name,balance\*1.05(σ balance≤5000(account))

**Null Values**

Tuplemay not have any values for some of the attributes. At that time the attribute is said

to have null value and is denoted by null. It simplifies an unknown value or a value does not

exist. The result of any arithmetic or comparison involving null is null. There are often more

than one possible way of dealing with null values, as a result our definition can sometimes

be arbitrary. Therefore arithmetic operations and comparison on null values should avoid if

possible.

**Advantages and limitation of relational algebra**