Introduction to lecture :

In this lecture, students will learn about different operations used in relational algebra :-

- Select operation
- Project operation
- Composition of relational operations
- Union operation
- Set difference operation

Relational Algebra :-

Relational algebra is a procedural query language that consists of a set of operations that take one or two relations as input and result into a new relation as an output.

These operations are divided into two groups -

One group consists of operations developed specifically for relational database such as select, project, rename, join and division.

The other group include set-oriented operations such as union, intersection, difference and cartesian product.

Some of the queries which are mathematical in nature, can not be expressed using the basic operations of the relational algebra. For such queries additional operations like aggregate functions and grouping are used such as finding sum, average, etc.

Tables :-

account-number	branch-name	balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

Fig: The account relation

branch-name	branch-city	assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

Fig: The branch relation

Customer-name	account-number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Fig: The depositor relation

loan-number	branch-name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

Fig: The loan relation

Customer-name	Customer-street	customer-city
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

Fig: The customer relation

Customer-name	loan-number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

Fig: The borrower relation

1. Select operation :- The select operation retrieves all those tuples from a relation that satisfy a specific condition.

→ The select operation can be viewed as the horizontal subset of a relation.

→ The greek letter sigma (σ) is used as a select operation.

→ In general, the select operation is specified as:

$$\sigma_{<\text{select-condition}>}(R)$$

where

selection-condition = the condition on the basis of which subset of tuples is selected.

2. The project operation :-

A relation might consists of a number of attributes, which are not always required.

→ The project operation is used to select some required attributes from a relation while discarding the other attributes.

→ It can be viewed as the vertical subset of a relation.

→ The greek letter pi (π) is used as project operator.

→ In general, the project operation is specified as

$$\pi_{<\text{attribute-list}>}(R)$$

Where, attribute-list = list of required attributes separated by comma

R = the name of relation

For ex-

write the query to list all loan number and the amount of the loan as :

$\Pi_{loan\text{-}number, amount}(loan)$

Result →

loan-number	amount
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500

3. Composition of relational operations :-

Consider the more complicated query "find those customers who live in Harrison". We write -

$\Pi_{customer\text{-}name}(\sigma_{customer\text{-}city = \text{"Harrison"}}(Customer))$

Notice that, instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

In general, since the result of the relational algebra operations can be composed together into a relational algebra expression.

4. The Union operation :—

The union operation, denoted by 'U', returns a third relation that contains tuples from or either of the operand relations. Consider two relations R1 & R2, their union is denoted as R1 U R2, which returns a relation containing tuples from both or either of the given relation.

For ex— Consider a query to find the names of all bank customers who have either an account or a loan or both.

→ To answer this query, we need the information in the depositor relation and in the borrower relation.

→ Names of all customers with a loan in the bank

$$\Pi_{\text{customer-name}}(\text{borrower})$$

→ To answer the query, we need the union of these two sets : we find this data by the binary operation, union denoted as U, so the

expression is –

$$\Pi_{customer\text{-}name}(borrower) \cup \Pi_{customer\text{-}name}(depositor)$$

→ A union operation r ∪ s to be valid, we require that two condition hold,

(1) → The relation r & s must be of the same arity, i.e., they must have the same number of attributes.

(2) → The domain of the ith attribute of r & ith attribute of s must be the same, for all i.

Result →

Customer-name
adams
curry
hayes
jackson
jones
smith
williams
windsay
johnson
turner

5. The set difference operation :- The set difference operation denoted by '−', allows us to find tuples that are in one relation but are not in another. The expression r − s produces a relation containing those tuples in r but not in s.

For ex- we can find all customers of the bank who have an account but not a loan by writting.

$$\Pi_{customer\text{-}name}(depositor) - \Pi_{customer\text{-}name}(borrower)$$

Conclusion of lecture :- In this lecture, we have concluded about some fundamental relational algebra operations like - select, project, union & set difference operation.

Introduction to lecture :-

In this lecture, we will discuss about some more additional relational algebra operations like -

- Natural Join
- Division Operation
- Assignment Operation
- Extended Relational Algebra Operations
  - Generalized Projection
  - Aggregate Functions
  - Outer Join
  - Null values

Natural Join :-

The natural join is a binary operation that allows us to combine certain selections and a cartesian product into one operation. It is denoted by the 'join symbol' ($\bowtie$).

The natural join operation forms a cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes.

For exc-

→ Find the names of all customers who have a loan at the bank, and find the amount of the loan.

$\Pi_{customer\text{-}name,\ loan\text{-}number,\ amount}$ (borrower ⋈ loan)

Since the schemas for borrower and loan have the attribute loan-number in common, the natural join operation considers only pair of tuples that have the same value on loan-number.

Result:-

Customer-name	loan-number	amount
Adams	L-16	1300
Curry	L-93	500
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-17	1000
Smith	L-23	2000
Smith	L-11	900
Williams	L-17	1000

→ Find the names of all branches with customers who have an account in the bank and who live in Harrison.

$\Pi_{branch\text{-}name}$ ($\sigma_{customer\text{-}city = \text{"Harrison"}}$ (Customer ⋈ account ⋈ depositor))

Result:

branch-name
Brighton
Perryridge

The theta join operation is an extension to the natural join operation that allows us to combine a selection and a cartesian product into a single operation.

Consider relations $r(R)$ and $s(S)$, and let θ be a predicate on attributes in the schema RUS. The theta join operation $r \bowtie_\theta s$ is defined as -

$$r \bowtie_\theta s = \sigma_\theta (r \times s)$$

For ex - Students $\bowtie_{S\#=200}$ Courses [Conditioned selection from r]

The Division Operation :-

The division operation, denoted by ÷, is suited to queries that include the phrase "for all".

Suppose that we wish to find all customers who have an account at all the branches in Brooklyn by the expression -

$$r1 = \Pi_{branch\text{-}name} (\sigma_{branch\text{-}city = "Brooklyn"} (branch))$$

Result:

branch-name
Brighton
Downtown

We can find all (customer-name, branch-name) pairs for which the customer has an account at a branch by -

$r_2 = \Pi_{customer\text{-}name,\ branch\text{-}name}\ (depositor \bowtie account)$

Result:

Customer-name	branch-name
Hayes	Perryridge
Johnson	Downtown
Johnson	Brighton
Jones	Brighton
Lindsay	Redwood
Smith	Miamus
Turner	Round Hill

Now, we need to find customers who appear in r_2 with every branch name in r_1. The operation that provides exactly those customers is the divide operation.

The query expression is -

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

The result of this expression is a relation that has the schema (customer-name) and that contains the tuple (Johnson)

→ Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$; i.e., every attribute of schema S is also in schema R. A tuple t is in $r \div s$ if and only if both of two conditions hold:

(1) t is in $\Pi_{R-S}(r)$

(2) for every tuple t_s in S, there is a tuple t_r in R satisfying both of the following:

a) $t_r[S] = t_s[S]$

b) $t_r[R-S] = t$

The Assignment Operation :-

The assignment operation, denoted by '←', works like assignment in a programming language.

For ex- to illustrate this operation, consider the definition of division, we could write $r \div s$ as :-

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

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

$result \leftarrow temp1 - temp2$

With the assignment operation, a query can be written as a sequential program consisting

of a series of assignments followed by an expression whose value is displayed as the result of the query.

For relational-algebra queries, assignment must always be made to a temporary relation variable. Assignments to permanent relations constitute a database modification.

Extended Relational Algebra Operations :-

The basic relational algebra operations have been extended in several ways. A simple extension is to allow arithmatic operations as part of projection & many more.

Generalized Projection :-

The generalized projection operation extends the projection operation by allowing arithmatic functions to be used in the projection list.

The generalized projection operation has the form -

$$\Pi_{F1, F2, \ldots Fn}(E)$$

where E is any relational algebra expression, and each of $F1, F2 \ldots Fn$ is an arithmatic expression involving constants and attributes in the schema of E.

For ex- Suppose we want to find how much more each person can spend (as expenses), we can write the following expression:

$$\Pi_{customer\text{-}name,\ limit - credit\text{-}balance}\ (credit\text{-}info)$$

The resulting attribute from the expression limit - credit-balance does not have a name, so we can apply the rename operation to the result

$$\Pi_{customer\text{-}name,\ (limit - credit\text{-}balance)\ as\ credit\text{-}available}\ (Credit\text{-}info)$$

Customer-name	limit	credit-balance
Curry	2000	1750
Hayes	1500	1500
Jones	6000	700
Smith	2000	400

Fig: Credit-info relation

Result of the expression:

Customer-name	Credit-available
Curry	250
Jones	5300
Smith	1600
Hayes	0

Fig: Result relation

Aggregate Functions :- Aggregate functions takes a collection of values and return a single value as a result.

There are many types of aggregate functions like sum, count, avg, min, max, etc.

The collections on which aggregate functions operate can have multiple occurences of a value; the order in which the values appear is not relevant. Such collections are called multi-sets. Sets are a special case of multisets where there is only one copy of each element.

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

$$G_1, G_2, \ldots G_n \; , \; F_1(A_1), F_2(A_2) \ldots F_m(A_m) \; E$$

where

E = any relational-algebra expression

$G_1, G_2 \ldots G_n$, constitute a list of attributes on which to group

F_i = an aggregate function

A_i = an attribute name

The tuples in (a group) the result of expression E are partitioned into groups in such a way that:

(1) All tuples in a group have the same values for $G_1, G_2, \ldots G_n$

(2) Tuples in different groups have different values for $G_1, G_2, \ldots G_n$.

employee-name	branch-name	salary
Adams	Perryridge	1500
Brown	Perryridge	1300
Gopal	Perryridge	5300
Johnson	Downtown	1500
Loreena	Downtown	1300
Peterson	Downtown	2500
Rao	Austin	1500
Sato	Austin	1600

Fig: The pt-works relation

→ Find out the total sum of salaries of all the part-time employees in the bank.

G_{sum} (salary) (pt-works)

→ Find the member of branches appearing in the pt-works relation

branch-name G_{sum} (salary) (pt-works)

Outer Join :-

The outer join operation is an extension of the join operation to deal with missing information.

For ex- consider the table which contain data on full time employees. Suppose that we want to generate a single relation with all the information about full-time employees.

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

Fig: employee relation

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

ft-works relation

Natural join operation is as:

employee ⋈ ft-works

Result:-

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

Fig: Result of employee ⋈ ft-works

In this result, we have lost some tuple's information like, street & city of Smith, branch-name & salary about ~~Gates~~, Gates.

We can use outer join operation to avoid this loss of information.

There are 3 forms of outer join -

1) left outer join ⟕
2) right outer join ⟖
3) full outer join ⟗

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

Fig: Result of 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
Gates	null	null	Redmond	5300

Fig: Result of 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

Fig: Result of employee ⟗ ft-works

Null values :-

The null value indicates "value unknown or non-exist", any airthmatic operations (such as +, −, *, /) incolving null values must return a null result.

Any comparisons such as (<, <=, >, >=, ≠) incolving a null value evaluate to special value 'unknown'.

Example of Division :-

Sno	Pno
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

A

Sno
S1
S2
S3
S4

A/B1

Sno
S1
S4

A/B2

Sno
S1

A/B3

Pno
P2

B1

Pno
P2
P4

B2

Pno
P1
P2
P4

B3

Conclusion of lecture :-

In this lecture, we have concluded about Natural join, Division operation, Assignment operation & some extended relational-algebra operations, such as, Generalized Projection, Aggregate Function, Outer join & Null values.

Introduction to lecture :

In this lecture we will discuss about following topics –

- Cartesian Product Operation
- Rename Operation
- Additional Relational Algebra Operation
- Set Intersection Operation

Cartesian Product Operation :–

The cartesian product, also known as cross product or cross join, returns a third relation that contains all possible combinations of the tuples from the two operand relations. The cartesian product is denoted by the symbol 'X'. The cartesian product of the relation R_1 and R_2 is denoted by $R_1 \times R_2$.

→ Since the same attribute name appear in both R_1 & R_2, we need to device a naming schema to distinguish between these attributes. We do so here by attaching to an attribute the name of the relation from which the attribute originally came.

For ex– The relation schema for.

r = borrower × loan is –

(borrower.customer-name, borrower.loan-number, loan.loan-number, loan.branch-name, loan.amount)

For ex -

Suppose that we want to find the names of all customers who have a loan at the Perryridge branch. We need the information in both the loan relation and the borrower relation to do so.

So, the query is -

$$\sigma_{borrower.loan\text{-}number = loan.loan\text{-}number}(\sigma_{branch\text{-}name = \text{"Perryridge"}}(borrower \times loan))$$

we get only those tuples of borrower × loan that pertain to customers who have a loan at the Perryridge branch.

Finally, since we want only customer-name, we do a projection:

$$\Pi_{customer\text{-}name}(\sigma_{borrower.loan\text{-}number = loan.loan\text{-}number}(\sigma_{branch\text{-}name = \text{"Perryridge"}}(borrower \times loan)))$$

2. The Rename operation :-

When relational algebra operations are performed, the resultant relations are unnamed, hence can not be used for later reference. In addition, it might be required to apply more operations on the relation obtained from other operations.

→ The Rename operation is used to provide name to relation obtained after applying an relational algebra operation.

→ The greek letter rho (ρ) is used as rename operator.

→ Relation obtained from any operation can be renamed by -

$$\rho(R, E)$$

where

ρ = rename operator

E = expression representing relational algebra

R = name given to the relation

Additional Relational Algebra Operations :

1) Set - intersection operation :-

Set-intersection is not a fundamental operation & does not add any power to the relational algebra.

For ex-

Suppose that we wish to find all customers who have both a loan and an account. Using set-intersection we can write -

$\Pi_{customer\text{-}name}(borrower) \cap \Pi_{customer\text{-}name}(depositor)$

Result –

Customer-name
Hayes
Jones
Smith

We can rewrite any relational algebra (operation) expression that uses set-intersection by replacing the intersection operation with a pair of set-difference operations as:

$$r \cap s = r - (r - s)$$

Conclusion of lecture :-

In this lecture, we have concluded about cartesian product, rename and set-intersection operation.

Introduction to lecture :-

In this lecture, we will discuss about the relational calculus, including following topics -

- Introduction to relational calculus
- Tuple relational calculus
  - Formal definition
  - Example queries.

Introduction to relational calculus :-

Relational calculus can define the information to be retrieved. It is not having any specific series of operations.

→ It comes in two flavours :
- Tuple Relational Calculus (TRC)
- Domain Relational Calculus (DRC)

→ Calculus has variables, constants, comparison operations, logical connectives and quantifiers.

→ It gives a way of referring to attributes of relations. (and the specific tuple in them).

→ It is based on predicate logic - gives the usual quantifiers to construct complex queries.

→ Expressions in the calculus are called formulas. An answer tuple is essentially an assignment

of constants to variables that make the formula evaluate to true.

TRC : Variables range over (i.e., get bound to) tuples.

DRC : Variables range over domain elements (= field values).

→ Both TRC & DRC are simple subsets of first-order logic

Tuple Relational Calculus :-

The tuple relational calculus, is a non-procedural query language.

→ It describes the desired information without giving a specific procedure for obtaining that information.

→ A query in the tuple relational calculus is expressed as -

$$\{ t \mid P(t) \}$$

where

$t \rightarrow$ tuple variable

$P(t) \rightarrow$ is a formula

It is the set of all tuples t such that predicate P is true for t.

- We use $t[A]$ to denote the value of tuple t on attribute A, and we use $t \in r$ to denote that tuple t is in relation r.

Example Queries :→

- If we want to find the branch-name, loan-number, and amount for loans of over $1200:

$$\{t \mid t \in loan \wedge t[amount] > 1200\}$$

- If we want only the loan-number attribute, we can write the query "Find the loan-number for each loan of an amount greater than $1200" as –

$$\{t \mid \exists s \in loan\ (t[loan\text{-}number] = s[loan\text{-}number] \wedge s[amount] > 1200)\}$$

- Consider the query "Find the names of all customers who have a loan from the Perryridge branch"

It requires two "there exists" clauses in TRC expression, connected by and ($\wedge$).

The query is as follows –

$\{t \mid \exists s \in borrower (t[customer\text{-}name] = s[customer\text{-}name] \wedge \exists u \in loan (u[loan\text{-}number] = s[loan\text{-}number] \wedge u[branch\text{-}name] = \text{"Perryridge"}))\}$

This expression is "The set of all (customer-name) tuples for which the customer has a loan that is at the Perryridge branch".

Result :-

Customer-name
Adams
Hayes

Formal Definition :-

A TRC is of the form

$\{t \mid P(t)\}$

Several tuple variables may appear in a formula. A tuple variable is said to be a free-variable unless it is quantified by a $\exists$ or $\forall$. Thus in

$t \in loan \wedge \exists s \in customer (t[branch\text{-}name] = s[branch\text{-}name])$

$\Rightarrow$ t is a free variable. Tuple variable s is said to be a bound variable.

Limitations of TRC :-

→ It can not express queries involving
- aggregations
- closure

→ It cannot express non-query operations
- insert
- delete
- update

because those involve modeling the changes in the state of a database.

Conclusion of lecture :-

In this lecture, we have concluded about the relational calculus and the definition, example queries and limitations of Tuple Relational Calculus.

Introduction to lecture :-

In this lecture, we will discuss about the second type of the relational calculus, including -

- Formal definition of DRC
- Example queries

Domain Relational Calculus :-

A second form of relational calculus, called Domain Relational Calculus, uses domain variables that take on values from an attributes domain, rather than values for an entire tuple.

Domain Relational Calculus serves as the theoritical basis of the widely used QBE language, just as relational algebra serves as the basis for the SQL language.

Formal definition :-

An expression in the DRC is of the form

$$\{\langle x_1, x_2, \ldots, x_n \rangle \mid P(x_1, x_2, \ldots, x_n)\}$$

where $x_1, x_2, \ldots, x_n$ represent domain variables

P represents a formula composed of atoms, as in TRC.

→ DRC query has the form:

$$\{ \{\langle x_1, x_2, \ldots x_n \rangle \mid P(\langle x_1, x_2, \ldots x_n \rangle)\}$$

Answer includes all tuples $\langle x_1, x_2, \ldots x_n \rangle$ that make the formula $P(\langle x_1, x_2, \ldots x_n \rangle)$ be true.

Formula is recursively defined, starting with simple atomic formulas (getting tuples from relations or making comparisons of values), and building bigger and better formulas using the logical connectives.

Example Queries :—

- Find the loan number, branch name and amount for loans of over $1200:

$$\{ \langle l, b, a \rangle \mid \langle l, b, a \rangle \in loan \wedge a > 1200 \}$$

- Find all loan numbers for loans with an amount for loan greater than $1200:

$$\{ \langle l \rangle \mid \exists b, a \, (\langle l, b, a \rangle \in loan \wedge a > 1200) \}$$

- Find the names of all customers who have a loan from the Perryridge branch and find the loan amount:

$$\{ \langle c, a \rangle \mid \exists l \, (\langle c, l \rangle \in borrower \wedge \exists b \, (\langle l, b, a \rangle \in loan \wedge b = \text{"Perryridge"})) \}$$

- Find the names of all customers who have a loan, an account, or both at the Perryridge branch:

$$\{ \langle c \rangle \mid \exists l \, (\langle c, l \rangle \in borrower \wedge \exists b, a \, (\langle l, b, a \rangle \in loan \wedge b = \text{"Perryridge"})) \vee \exists a \, (\langle c, a \rangle \in depositor \wedge \exists b, n \, (\langle a, b, n \rangle \in account \wedge b = \text{"Perryridge"})) \}$$

- Find the names of all customers who have an account at all the branches located in Brooklyn:

$$\{ \langle c \rangle \mid \exists n \, (\langle c, n \rangle \in customer) \wedge \forall x, y, z \, (\langle x, y, z \rangle \in branch \wedge y = \text{"Brooklyn"} \Rightarrow \exists a, b \, (\langle a, x, b \rangle \in account \wedge \langle c, a \rangle \in depositor)) \}$$

In english, we interpret this expression as "The set of all (customer-name) tuples c such that, for all (branch-name,

branch-city, assets) tuples, x, y, z, if the branch city is Brooklyn, then the following is true":

- There exist a tuple in the relation account with account number a and branch name x

- There exist a tuple in the relation depositor with customer c and account number a.

Conclusion of lecture:

In this lecture, we have concluded about the formal definition and example queries of DRC.