DEPARTMENT OF COMMERCE (CA)
DATABASE MANAGEMENT SYSTEM (Semester-III)
II B.COM(CA) Sub Code-18BCA32C
UNIT –II

Relational Approach: Relational Data Structure : Relation, Domain, Attributes, Key Relational Algebra - Introduction, Traditional Set Operation. Attribute names for derived relations - Special Relational Operations.

Relational Approach

Introduction:

The relational model has established itself as the primary data model for commercial data-processing applications. The first database systems were based on either the network model or the hierarchical model. The relational model is now being used in numerous applications outside the domain of traditional data processing.

Structure of relational databases.

A relational database consists of a collection of tables, each of which is assigned a unique name. A row in a table represents a relationship among a set of values. The rows are termed as tuples and columns are termed as attributes. Since a table is a collection of such relationships, there is a close correspondence between the concept of table and the mathematical concept relation, from which the relational data model takes its name.

The following account table or relation has three column headers: branch-name, account-number and balance. These are the attributes (columns are referred as attributes). For each attribute there is a set of permitted values, called the domain of that attribute. For the attribute, branch-name set of all branch-names is its domain.

The account relation

Branch-name	Account-number	Balance
Downtown	A-101	500
Mianus	A-215	700
Perry ridge	A-102	400
Round Hill	A-305	350
Brighton	A-201	900
Redwood	A-222	700
Brighton	A-217	750

Let D1 denote the set of all branch-names, D2 denote the set of all account-numbers, and D3 the set of all balances. In the account relation it consists of a 3-tuple (v1, v2, v3), were v1 is a branch name, v2 is an account number and v3 is a balance. The account will contain only a subset of the set of all possible rows. It can be represented as

D1 * 2 * D3

In general a table of n attributes must be a subset of

$$D1 * D2 * \ldots\ldots D_{n-1} * D_n$$

The relation is said to be a subset of a Cartesian product of a list of domains. Tables are relations and the mathematical terms relation and tuple is used for the terms table and row respectively. In the account relation of the above figure there are seven tuples. Let the tuple variable t refer to the first tuple of the relation .We use the notation t [branch-name] to denote the value of t on the branch-name attribute. Thus, t [branch-name]="Downtown", and t [balance]=500.Since the relation is a set of tuples, we use the mathematical notation of t E r to denote that tuple r is in relation r.

Domain: - Domain is a pool of values.
Also we can say that domain is atomic if elements of the domain are considered to be individual units. For example, the set of integers is a nonatomic domain. The distinction is that we do not normally consider integers to have subparts, but we consider sets of integers to have subparts-namely, the integers comprising the set. It is possible for several attributes to have the same domain.

The customer relation

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

It is possible for several attributes to have the same domain. For example, suppose that we have a relation customer that has the three-attribute customer-name, customer-street and customer-city, and a relation employee that includes the attribute employee-name. It is possible that the attributes customer-name and employee-name will have the same domain: the set of all person names. The domains of balance and branch-name are certainly distinct. It is perhaps less clear whether customer-name and branch-name should have the same domain. At the physical level, both customer names and branch-names are character strings. However, at the logical level, we may want customer-name and branch-name to have distinct domains.

Relation:

Definition for relation (mathematically):

Given a collection of set D1, D2,……Dn (not necessarily distinct,R is a relation on those n sets if it is a set of ordered n-tuples <d1,d2,……dn> such that d1 belongs to D1,d2 belongs to D2 ,…..dn belongs to Dn.Set D1,D2,D3,…..Dn are the domains of R.The value of n is the degree of R.

The concepts of relation correspond to the programming-language notion of a variable. The concept of a relation schema corresponds to the programming-language notion of type definition. It is convenient to give a name to a relation schema, just as we give names to type definitions in programming languages. We adopt the convention of using lowercase names for relations, and names beginning with an uppercase letter for relation schemas. For example,

Account-schema=(branch-name, account-number, balance)

The explanation of relation can be expressed diagrammatically with the help of E-R diagrams. Before discussing E-R diagrams, the common terms used in the diagrams is analysed.

Entity: This is a thing or object in the real world that is distinguishable from all other objects. For example, each person in an enterprise is an entity. An entity has a set of properties, and the values for some set of properties may uniquely identify entity. For example, the social-security number 677-89-9011(employee number 1111) uniquely identifies one particular person in the enterprise.

Entity Set: An entity set is a set of entities of the same type that share the same properties or attributes. The set of all persons who are customers at a given bank, for example, can be defined as the entity set customer.

Attributes: An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set. Possible attributes of customer entity are customer-number, customer-street, and customer-city. The following attribute types, as used in the E-r model, can characterize an attribute.

- **Simple and Composite attributes:** The attributes, which can be divided into subparts, are composite attribute. For example, name is an attribute, which is combination of first-name, middle name, and last-name.

- **Single-valued and Multivalued attributes:** The attributes that we have specified in our examples all have a single value for a particular entity. For instance, the loan-number attribute for a specific loan entity refers to only one loan number. Such attributes are said to be single valued. There may be instances where an attribute has a set of values for a specific entity.

- **Null attributes:** A null value is used when an entity does not have a value for an attribute.

- **Derived attribute:** The value for this type of attribute can be derived from the values of other related attributes or entities. For instance, let us say that the customer entity set has an attribute loans-held, which represents how many

loan a customer entity set has from the bank. We can derive the value for this attribute by counting the number of loan entities associated with that customer.

Relationship sets

Consider the relation loan.

Branch-name	Loan-number	Amount
Downtown	L-17	1000
Redwood	L-23	2000
Perry ridge	L-15	1500
Downtown	L-14	1500
Mianus	L-93	500
Round Hill	L-11	900
Perry ridge	L-16	1300

A relationship is an association among several entities. For example, we can define a relationship that associates customer Hayes with loan number L-15.This relationship specifies that Hayes is a customer with loan number L-15.

A relationship set is a set of relationships of the same type.Formally.it is a mathematical relation on n>=2 (possibly non distinct) entity sets. If E1, E2,…..En are entity sets, then a relationship set R is a subset of

{(e1, e2,……..,en)|e1 ∈ E1,e2 ∈ E2 ,…..en ∈ En}

Where (e1, e2,…….en) is a relationship.

Consider the two entity sets customer and loan, we can define the relationship set borrower to denote the association between customers and the bank loans that the customers have. As another example, consider the two-entity sets loan and branch. We can define the relationship set loan-branch to denote the association between a bank loan and the branch in which that loan is maintained.

Each row of the table represents one n-tuple of the relation. The number of tuples in the relation is called the cardinality of the relation. Eg. The cardinality of the relation loan is 7.

The relations may be unary, binary, ternary, n-ary etc.

Unary: Relations of degree one is unary.

For ex, the query Find the branch name that issued loan with number L-17.The output will be

Branch-name
Downtown

Binary: Relations of degree two are binary.

x, Find branch-name and amount for loan-number L-17 from branch relation

The output will be,

Branch-name	Amount
Downtown	1000

Ternary: Relations of degree three are ternary

N-ary: Relations of degree n are n-ary.

Mapping cardinalities: Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via relationship set. Mapping cardinalities are most useful in describing binary relationship sets, although occasionally they contribute to the description of relationship sets that involve more than two entity sets.
For binary relationship set R between sets A and B, the mapping cardinality must be one of the following:

One to one: An entity is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

One to Many: An entity in A is associated with any number of entities in B.An entity in B, however, can be associated with at most one entity in A.

Many to one: An entity in A is associated with at most one entity in B.An entity in B, however, can be associated with any number of entities in A.

Many to Many: An entity in A is associated with any number of entities in B, and an entity in B is associated with any number of entities in A.

Keys:

In a relation there is one attribute whose values is unique within the relation and thus can be used to identify the tuples of that relation.

For ex, in the above said loan relation the loan number can be considered as a key, which is unique, and can be used to distinguish all other tuples in that relation. Befrore discussing on various keys let us have a glance on integrity constraints.

Integrity constraints:

An integrity constraint is a mechanism used by oracle to prevent invalid data entry into the table. It is nothing but enforcing rule for the coloumn in a table. The following are the various types of integrity constraints: -

***Domain integrity constraints**

Maintains value according to the specification like ‘not null’ condition, so that the user has to enter a value for the coloumn on which it is specified.

‘Not null’ and ‘Check’ constraints fall unde this category.

***Entity integrity constraint**

Maintains uniqueness in a record.

***Referential integrity constraint**

Enforces relationship between tables

To establish a ‘parent-child’ or a ‘master-detail’ relationship between two tables having a common column we make use of referential integrity constraints. To implement this we should define the column in the parent table as a primary key and the same column in the child table as a foreign key referring to the corresponding parent entry.

We define constraint to either at table or column level. If it is defined at the table level, then it can be enforced to any number of columns in a table .On other hand, if it is defined at the column level then it holds good only for the column for which it is defined.

Various keys related to relational approaches are

Primary Key: Primary key is a set of one or more attributes that, taken collectively allows us to identify uniquely an entity in the entity-set.

Ex.1) An-number in the loan relation
2) Also the combination of branch-name and loan-number

Candidate Key: Several distinct sets of attributes could serve as candidate key

Referenced key: It is a unique or a primary key, which is defined on a coloumn belonging to the parent table.

Foreign Key: A coloumn or combination of coloumns included in the definition of referential integrity, which would refer to a referenced key.

Child table: This table depends upon the values present in the referenced key of the parent table, which is referred by a foreign key.

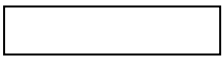
Parent table: This table determines whether insertion or updation of data can be done in child table. This table would be referred by child table’s foreign key.

On delete cascade clause

If all rows under the referenced key coloumn in a parent table are deleted, than all rows in the child table with dependent foreign key will also be deleted automatically.

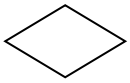
Entity-Relationship Diagrams:

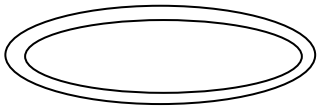
An E-R diagram can express the overall logical structure of a database graphically. Such a diagram consists of the following major components:

The symbol used to represent entity is rectangle 

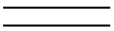
The symbol used to represent attribute is ellipse 

The symbol used to represent links is lines

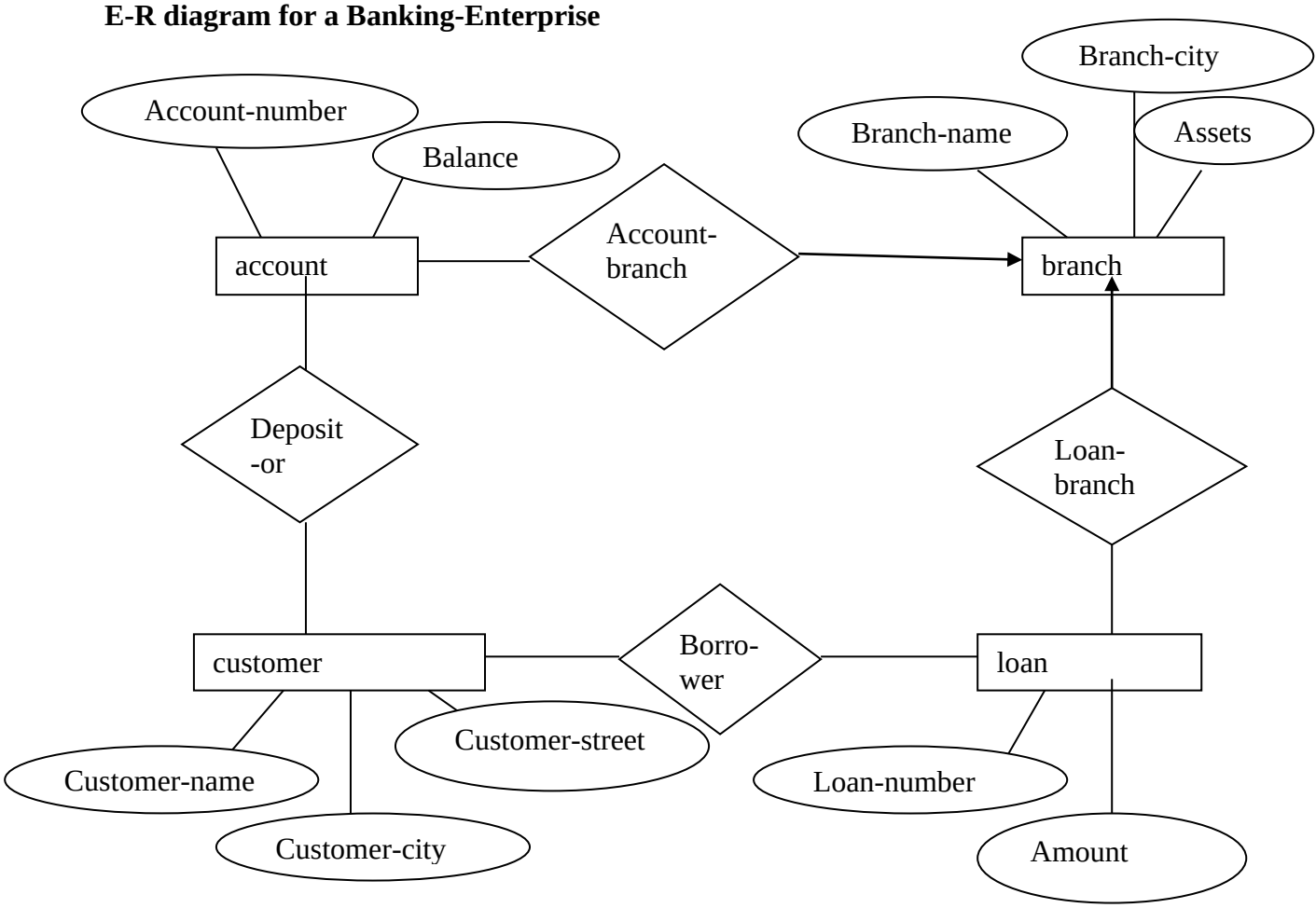
The symbol used to represent the relation is 

The symbol used to represent multivalued attributes is Double ellipses 

The symbol used to represent the derived attributes is dashed ellipses 

The symbol used to represent the total partition of entity in a relationship set is double lines. 

E-R diagram for a Banking-Enterprise



Various relations used for the discussion of this chapter are
1.Account relation

Branch-name	Account-number	Balance
Downtown	A-101	500
Mianus	A-215	700
Perry ridge	A-102	400
Round Hill	A-305	350
Brighton	A-201	900
Redwood	A-222	700
Brighton	A-217	750

2.Loan relation

Branch-name	Loan-number	Amount
Downtown	L-17	1000
Redwood	L-23	2000
Perry ridge	L-15	1500
Downtown	L-14	1500
Mianus	L-93	500
Round Hill	L-11	900
Perry ridge	L-16	1300

3.Branch relation

Branch-name	Branch-city	Assets
Downtown	Brooklyn	9000000
Redwood	Palo alto	2100000
Perryridge	Horse neck	1200000
Mianus	Horse neck	400000
Round hill	Horse neck	8000000
Pownal	Bennington	300000
North town	Rye	3700000
Brighton	Brooklyn	7100000

4.Customer relation

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

5.Depositor relation

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

6.Borrower relation

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

Relational Algebra

Note: Query languages

A query language is a language in which a user requests information from the database. These languages are typically of a level higher than that of a standard programming language. Query languages can be categorized as being either procedural or non-procedural .In procedural language, the user instructs the system to perform a sequence of operations on the database to compute the desired result. In a non-procedural language, the user describes the information desired without giving a specific procedure for obtaining that information.

Introduction

Relational algebra is a collection of operations on relations. Also it is a procedural query language, it consists of a set of operations that take one or two relations as input and produce a new relation as their result.

The fundamental operations or traditional set operations available with relational algebra are select, project, set difference, Cartesian, rename, union. In addition to the fundamental operations, there are several other operations-namely, set intersection, natural join, division, and assignment. These operations will be defined in terms of the fundamental operations. Also we can state the selection, projection, join and division operations as special relational operators.

Fundamental operations

The select, project and rename operations are called unary operations, because they operate on one relation. The other three operations union, setdifference and Cartesian product operate on pairs of relations and are, therefore called binary operations.

The select operation

The select operation selects tuples that satisfy a given predicate. The lowercase Greek letter sigma (σ) is used to denote selection. The predicate appear as a subscript to σ. The argument relation is given in parenthesis following the σ.

Example:
1.Select those tuples of the loan relation where the branch is "Perryridge".

$\sigma_{branch\ \_name="perryridge"}(loan)$

The result of the query is

Branch-name	Loan-number	Amount
Perryridge	L-15	1500
Perryridge	L-16	1300

2.Find all tuples in which the amount lent is more than $1200

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

All comparisons using =,≠, <,≥,≥ in the selection predicate. Also we can combine larger predicates using the connectives and (^) and or (ᵛ).

3.Find those tuples pertaining to loans of more than $1200 made by Perryridge branch

$\sigma_{branch\ \_name="perryridge"\wedge amount>1200}(loan)$

The project operation

Suppose we want to list all loan numbers and the amount of the loans, but do not care about the branch name. The project operation allows us to produce this relation. The project operation is a unary operation that returns its argument relation, with certain attributes left out. Since a relation is a set, any duplicate rows are eliminated. Projection is denoted by the Greek letter pi (π). We list those attributes that we wish to appear in the result as subscript to π.The argument relation follows in parentheses.

Example:
1.List all loan numbers and the amount of the loan .The corresponding query is

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

The relation that results from this query is

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

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 results in a relation containing those tuples in r but not in s.

Example:

1.Find all customers of the bank who have an account but not a loan

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

The result will be

Customer-name
Johnson
Turner
Lindsay

For a set difference operation r-s to be valid, we require that the relations r and s be of the same arity, and that the domains of the i^{th} attribute of r and the i^{th} attribute of s be the same.

The Cartesian – product operation

The Cartesian-product operation, denoted by a cross (X), allows us to combine information from any two relations. We write the Cartesian product of relations r1 and r2 as r1 X r2. Since the same attribute name may appear in both r1 and r2, we need to devise 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 example, the relation schema for r = borrower X loan is

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

So now we can distinguish borrower.loan-number from loan.loan-number.For those attributes that appear in only one of the two schemas,we shall usually drop the relation-name prefix.We can wrte the relation schema for r as

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

This above naming convention requires that the relations that are arguments of the Cartesian-product operation have distinct names.

Assume that we have n1 tuples in borrower and n2 tuples in loan. Then, there are n1 * n2 ways of choosing a pair of tuples –one tuple from each relation; so there are n1*n2 tuples in r. In particular ,note that for some tuples t in r,it may be that t[borrower. loan-number] not equal to t[loan.loan-number].

In general ,if we have relations r1(R1) and r2(R2),then r1 X r2 is a realtion whose schema is the concatenation of R1 and R2.Relation R contains all tuples t for which there is a tuple t1 in r1,and t2 in r2 for which t[R1]=t1[R1] and t[R2]=T2[R2].

For example

1.if 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.If we write

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

Customer-name	Borrower.loan-nu	Branch-name	Loan.loan-numb	Amount
Jones	L-17	Downtown	L-17	1000
Jones	L-17	Redwood	L-23	2000
…….	…….	…….	……..	…..
…….	…….	……	…….	…..
…….	…….	……	…….	…..
Adams	L-16	Round hill	L-11	900
Adams	L-16	Perryridge	L-16	1300

Table:Result of borrower X loan

Now the output of the query stated above will be as

Customer-name	Loan-number	Branch-name	Loan-number	Amount
Jones	L-17	Perryridge	L-15	1500
Jones	L-17	Perryridge	L-16	1300
Smith	L-23	Perryridge	L-15	1500
Smith	L-23	Perryridge	L-15	1300
Hayes	L-15	Perryridge	L-15	1500
Hayes	L-15	Perryridge	L-16	1300
Jackson	L-14	Perryridge	L-15	1500
Jackson	L-14	Perryridge	L-16	1300
Curry	L-93	Perryridge	L-15	1500
Curry	L-93	Perryridge	L-16	1300
Smith	L-11	Perryridge	L-15	1500
Smith	L-11	Perryridge	L-16	1300
Williams	L-17	Perryridge	L-15	1500
Williams	L-17	Perryridge	L-16	1300
Adams	L-16	Perryridge	L-15	1500
Adams	L-16	Perryridge	L-16	1300

Table:result of query $\sigma_{\text{branch-name="Perryridge"}}$(borrower X loan)

The relation describes the details relating to perryridge branch alone.But there is a chance that many customers may not have a loan at perryridge branch.So the query can be re-written as

$$\sigma_{\text{borrower.loan-number=loan.loan-number}}$$

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

In order to retrieve only the customer-name ,we vcan have the projection operation as

$$\pi_{\text{customer-name}}(\sigma_{\text{borrower.loan-number}} = \text{loan.loan-number}$$

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

The result is as shown below

Customer-name
Hayes
Adams

Table:Result of $\pi_{\text{customer-name}}(\sigma_{\text{borrower.loan-number}} = \text{loan.loan-number}$

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

The rename operation

Unlike relations in the database, the results of relational-algebra expressions do not have a name that we can use to refer to them. It is useful to be able to give them names; the rename operator, denoted by the lower-case Greek letter rho (ρ), lets us perform this task.

Given a relational-algebra expression E, the expression

$$\rho_x(E)$$

returns the result of expression E under the name x.

A relation r by itself is considered to be a trivial relational-algebra expression. Thus, we can also apply the rename operation to a relation r to get the same relation under a new name.

A second form of the rename operation is as follows. Assume that a relational-algebra expression E has arity n. Then the expression

$$\rho_x(A1,A2,.....An)(E)$$

returns the result of expression E under the name x,and with the attributes renamed to A1,A2,.....An.

For example,

1.Find the largest balance in the bank

Steps invloved are

- Compute first the relation consisting of those balances that are not the largest
- The take the set difference between the relation balance(account)
- Then comes the temporary relation

The corresponding queries are

$$\pi_{account.balance}(\ \sigma_{account.balance\ <\ d.balance}(account\ X\ \rho_d\ (account)))$$

This expression gives those balances in the account relation for which a larger balance appears somewhere in the account relation(renamed as d).The result contains all balances except the largest one.
The relation is

Balance
500
700
400
350
750

The query to find the largest account balance in the bank can be written as follows:

$$\pi balance(account) -$$

$$\pi_{account.balance}\ (\sigma_{account.balance\ <d.balance}(account\ X\ \rho_d\ (account)))$$

the result of this query is

Balance

900

Fig: largest account balance in the bank

2.Find the names of all customers who live on the same street and in the same city as Smith
The street and city of smith can be obtained by writing as

$$\pi_{\text{customer-street,customer-city}}(\sigma_{\text{customer-name="Smith"}}(\text{customer}))$$

In order to find other customers with this street and city, we must reference the customer relation a second time. In the following query, we use the rename operation on the preceding expression to give its result the name smith-addr, and to rename its attributes to street and city, instead of customer-street and customer-city:

$$\pi_{\text{customer.customer-name}}(\sigma_{\text{customer.customer-street=smith-addr.street}\wedge\text{customer.customer-city=smith-addr.city}}(\text{customer} \times \rho_{\text{smith-addr(street,city)}}(\pi_{\text{customer-street,customer-city}}(\sigma_{\text{customer-name="Smith"}}(\text{customer})))))$$

The result of this query is as shown below

Customer-name
Smith
curry

Additional operations or special relational operations

1.The set-intersection operation
The symbol used to identify is $\cap$.

Example:
1.Find all customers who have both a loan and an account.
Query is

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

The result will be

Customer-name
Hayes
Jones
Smith

Table: customers with both an account and a loan at the bank

The intersection operation can be replaced using the set difference operation as
$r \cap s = r-(r-s)$

The Union operation

With the help of this operation we can choose the details which are present in either of two relations.

For example:

1.Find the names of all bank customers who have either an accoubt or a loan or both. The customer relaion does not contain the information ,since a customer does not need to have either an account or a loan at the bank.And to answer this query we need the information in the depositor relation and in the borrower relation .

*To find the customers with loan at the bank we use

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

*To find the names of all customers with an account in the bank:

$\pi_{\text{customer\_name}}(\text{depositor})$

To find both account and loan holding customers we need to union these two as

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

The result of this query is

Customer-name
Johnson
Smith
Hayes
Turner
Jones
Londsay
Jackson
Curry
Williams
Adams

For union operation r U s to be valid, we require two conditions:

1.The relations r and s must be of the same arity. That is, they must have the same number of attributes.
2.The domain of the ith attribute of r and the ith attribute of s must be the same, for all i.

Where r and s can be, in general temporary relations that are the result of relational-algebra expressions.

The natural-join operation

It is often desirable to simplify certain queries that require a Cartesian product. A query that involves a Cartesian product includes a selection operation on the result of the Cartesian product.

Assume:

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

Steps :

1.Form the Cartesian product of the borrower and loan relations.

2.Select those tuples that pertain to only the same loan-number.
3.Project the customer-name,loan-number and amount.

$\pi_{customer\text{-}name,loan.loan\text{-}number,amount}$

$(\sigma_{borrower.loan\text{-}number=loan.loan\text{-}number}(borrower \times loan))$

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 ⋈.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 example:
1.Find the names of all customers who have a loan at the bank, and find the amount of the loan.

$\pi_{customer\text{-}name,loan\text{-}number,amount}(borrower \bowtie loan)$

The result of the query is

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

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

$\pi_{branch\text{-}name}(\sigma_{customer\text{-}city=\text{"Harrison"}}(customer \bowtie account \bowtie depositor))$

The result of the query is

Branch-name
Brighton
Perryridge

The division operation

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

Example:
1.Find all customers who have an account at all the branches located in Brooklyn.
Steps:
1.All branches in Brooklyn can be obtained as

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

The result is

Branch-name
Brighton
Downtown

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

$$r2=\pi_{\text{customer-name,branch-name}}(\text{depositor}\bowtie \text{account})$$

Customer-name	Branch-name
Johnson	Downtown
Smith	Mianus
Hayes	Perryridge
Turner	Round hill
Williams	Perryridge
Lindsay	Redwood
Johnson	Brighton
Jones	Brighton

Table:Result of $\pi_{\text{customer-name,branch-name}}(\text{depositor}\bowtie \text{account})$

Our question is to find those customers who appear in r2 with every branch name in r1.We formulate the query by writing

π customer-name,branch-name(depositor $\bowtie$ account)

$\therefore$ πBranch-name(σbranch-city="Brooklyn"(branch))

Extended relational-algebra operations

The basic relational-algebra expressions have been extended in several ways. A simple extension is to allow arithmetic operations as part of projection. An important extension is to allow aggregate operations, such as computing the sum of the elements of a set, or their average. Another important extension is the outer-join operation, which allows relational-algebra expressions to deal null values, which model missing information.

Generalized Projection

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

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

Where E is any relational-algebra expression, and each F1, F2,…Fn are arithmetic expressions involving constants and attributes in the schema of E.As a special case, the arithmetic expression may be simply an arithmetic or a constant. The following example demonstrates the basis for the use of the generalized projection operation. Suppose we have a relation credit-info, as shown, which lists the credit limit and expenses so far .If we want to find how much more each person can spend, we can write the following expression:

πcustomer-name,limit - credit-balance(credit-info)

Customer-name	Limit	Credit-balance
Jones	6000	700
Smith	2000	400
Hayes	1500	1500
Curry	2000	1750

Table:The credit-info relation

Customer-name	Limit-credit_balance
Jones	5300
Smith	1600
Hayes	0
Curry	250

The result of πcustomer-name, limit - credit-balance (credit-info)

Outer join

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

Aggregate functions

Aggregate functions are functions that 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.

The function sum applied on the collection

<1,1,3,4,4,11>

returns the value 24.

The function avg returns the average of the values. So average of the above is 4.

The function count returns the number of the elements in the collection and would return 6 on the preceding collection.

The functions min and max, returns the minimum and maximum values in a collection; they return 1 and 11.

Examples:

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

The query is

Sum salary (pt-works)

The result of this query is a relation with a single attribute, containing a single row with a numerical value corresponding to the sum of all the salaries of all employees working part-time in the bank.

BOOKS FOR REFERENCE:

1. Database systems concepts by Abraham Silberschatz, Henry F. Korth.
2. An Introduction to Database System – C. Dsai.
3. An introduction to Database Systems (Seventh Edition) – C.J. Date

Prepared by Dr.N.SHANMUGAVADIVU