3.0 Relational Model

3.1 Relational Model

Relational database model is a primary data model for commercial data-processing applications. It is popular because of its simplicity; it provides simple but powerful way of representing data. It also supports complex query. Database in a relational model is simply a collection of one or more relations, where each relation is represented by table with rows and columns. It allows simple high level languages to query data.

3.2 Structure of Relational Databases

In relational model, table is a major construct for representing data. Relational database consist set of tables. A row in a table represents a relationship among set of values. So table can be refers as a collection of such relationship.

3.2.1 Basic Construct

To illustrate the basic structure of database, let us consider a table "account"

account_number	branch_name	balance
A-1	Kathmandu	500
A-2	Lalitpur	300
A-3	Kathmandu	700
A-4	Bhaktpur	600

Figure: The account table

The columns of table "account" are: account_number, brance_name and balance refer attributes. The set of permitted values for each attribute known as domain of that attribute. For example: set of all accounts numbers is a domain of attribute account_number.

Let D_1 denotes set of all account number, D_2 denotes set of all branch names and D_3 denotes set of all balances. Any row of table "account" must consist 3-tuple (v_1, v2, v3), where v_1 is account number (i.e. v_1 is a domain of D_1), v_2 is branch name (i.e. v_2 is a domain of D_2), and v_3 is account balance (i.e. v_3 is a domain of D_3). In general, we can express table "account" will contain only subset of all possible rows. That is, "account" is a subset of

$$D1 \times D_2 \times D_3$$

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

$$D_1 \times D_2 \times D_3 \times . . \times D_{n-1} \times D_n$$

This implies that definition of table is almost similar to the definition of relation in mathematics. In mathematics, relationship is a subset of Cartesian product of a list of domains. Therefore, in relational model, table can be refer as a relation and row of table can be refers as tuple. We can define tuple variable to represent a tuple. The above account relation consist seven tuples. Assume that tuple variable t represents first tuple of the relation. Then, the notation t[account_number] indicates value of t on account_number attribute. That is, t[account_number]="A-1". Similarly t[branch]="Kathmandu", and t[balance]=500. We can also represent it as follow: t[1] where 1 indicated first attribute of relation; that is "account_number". Therefore, t[1]="A-1". In relational model, we can also express relation as a set of tuples. So definitely, $t \in r$.

3.0 Relational Model

For each relations r, domains of all attributes of r must be atomic. The domain is said to be ***atomic*** if elements of domain is indivisible unit. Domains of multivalued and composite attributes are **nonatomic**.

Several attributes may have same domain. Suppose we have two relation customer with customer_name as one of its attribute and employee is another relation with one of its attribute as employee_name. It is possible that attribute customer_name and employee_name may have same domain. If we look attributes: customer_name and branch_name of relations customer and account respectively, at physical level, their domain may be same, both are defined by set of character string. But at logical level, customer_name and branch_name must have distinct domain. Null value is a member of any possible domain. It signifies value is unknown or does not exist. Domain for customer_phoneno of customer entity set may null, meaning is that particular customer does not have any phone number or phone no. is not available.

3.2.2 Database Schema

A relation schema is a list of attributes and their corresponding domains. For example, the relation schema for relation customer is express as

Customer-schema = (customer_id, customer_name, customer_city)

We may also specify domains of attributes as

Customer-schema = (customer_id: integer, customer_name: string, customer_city:string)

We may state customer is a relation on Customer-schema by

customer(Customer-schema)

Values or data contain in relation change when it is updated. Relation instance is a snapshot of data in relation at particular time. But, in general, we simply say relation even it is actually relation instance.

In terms of relation, relational database is a collection of relations and relational database schema is a collection of schemas for relations in database. It describes logical design of database. The instance of relational database is collection of relation instances. It is actually a snapshot of data in database at a particular time.

Database schemas for banking enterprise

Branch-schema = (branch_name,branch_city,assets)
Account_schema = (account_number,branch_name,balance)
Customer-schema = (customer_id,customer_name,customer_street,customer_city)
Depositor-schema = (customer_id,account_number)
Loan-schema = (loan_number,branch_name,amount)
Borrower-schema = (customer_id,loan_no)

3.2.3 Keys

In relational model, keys (superkey, candidate key and primary key) play important roles. For example: in Branch-schema, {branch_name} and {branch_name, branch_city} are superkey. Since {branch_name} itself is a superkey, {branch_name, branch_city} can not candidate key in Branch-schema. In Branch-schema, {branch_name} is a single candidate key so ultimately it is a primary key of Branch-schema.

Let R be a relation schema and $K \subseteq R$. If K is superkey for R then it restricts relations r(R) in which no two distinct tuples have same values on all attribute in K. That is, if t_1 and t_2 are in r and $t_1 \neq t_2$ then $t_1[K] \neq t_2[K]$.

Relational database schema can be derive from an E-R schema where primary key for relation schema is primary key of entity or relationship set from which relation schema is derived. If relation is derive from strong entity set then primary key for relation is primary key of that strong entity set. Similarly, if relation is derived from weak entity set then primary key for relation is union of primary key of strong entity set and discriminator of weak entity set. And the relation consist

- all attributes of weak entity set
- primary key of the strong entity set on which weak entity set depends

In relational model, relationship set is also represented by a relation. Its primary key is depends up on mapping cardinalities of relationship set. If relationship is many to many then primary key of relation representing relationship set is a union of primary keys of related entity sets. If relationship is one to one then primary key of relation representing relationship set is primary key of any one related entity set. Similarly, if relationship is many to one then primary key of relation representing relationship set is the primary key of the "many" entity set.

In relational model, one relational schema may contain primary key of another relation schema. If relation schema r_1.contains primary key of another relation schema r_2 then this attribute (i.e. PK in r2) in r_1 called *foreign key*. The relation r1 called referencing relation (detail table) of the foreign key dependency and r_2 called referenced relation (master table).

Example:
In Branch-schema, branch_name is a primary key. In Account-schema, branch_name is a foreign key referencing Branch-schema. This implies, in any database instance, any tuple t_a in account relation, there must be some tuple, t_b in branch relation such that the value of the branch_name attribute of t_a is same as the values of the primary key, branch_name of t_b..

3.2.4 Schema Diagram

Schema diagram is a graphical representation of database schema along with primary key and foreign key dependencies.
In schema diagram, each relation is represented by box where attributes are listed inside box and relation name is specified above it. Primary key in relation is place above the horizontal line that crosses the box. Foreign key in schema diagram appear as arrow from the foreign key attributes of the referencing relation to the primary key of the referenced relation.

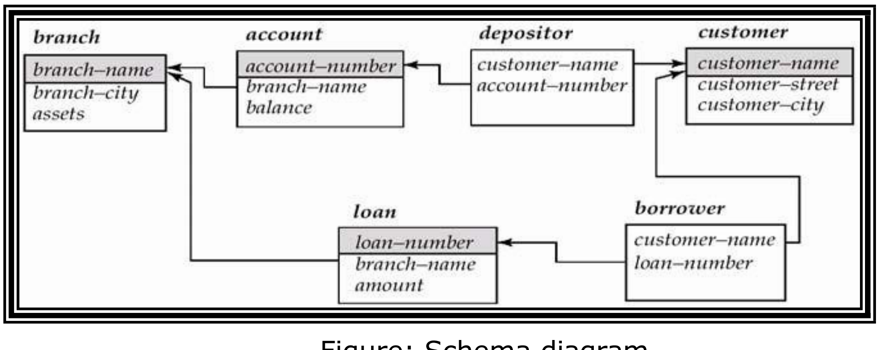

Figure: Schema diagram

3.0 Relational Model

Note: The difference between E-R diagram and schema diagram is, E-R diagram do not shows the foreign key but schema diagram shows it explicitly.

3.2.5 Query Languages

Query language is a language through which user request information from database. Query languages can be categories in procedural and non procedural. In procedural query language, user required to specify sequence of operations to system to compute desired information where as in nonprocedural query language, user need to specify required information without specifying special procedure for obtaining that information.

Most commercial relational database system offers query language, both procedural and non procedural. SQL is most popular nonprocedural query language.

In this section we discuss pure query language: relational algebra, tuple relational calculus and domain relational calculus. These query languages can not commercially use by people but it describes fundamental techniques for extracting data from database and provides basis for commercial query language.

3.3 Relational algebra

The relational algebra is a procedural query language. It consist set of operation that takes one or more relations as inputs and produce a new relation as output. The fundamental operations in relational algebra are selection, projection, union, set difference, Cartesian product, and rename. Set intersection, natural join, division and assignments other operations of relational algebra which can be define in terms of fundamental operations.

3.3.1 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 after σ. That is, general structure of selection is

$$\sigma_p(r)$$

where p is selection predicate.

Formally, selection operation define as

$$\sigma_p(r) = \{t | t \in r \text{ and } p(t)\}$$

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

<attribute>op<attribute>or <constant>

where op is one of the comparison operators: $=, \neq, <, \leq, >, \geq$

Examples:

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

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

output = {t | t[branch_name] = Kathmandu }

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

$$\sigma_{amount>5000}(loan)$$

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

$$\sigma_{branch\_name="Kathmandu" \wedge amount>5000}(loan)$$

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

The general structure of projection is

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

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

Example:Find account number and their balance from account relation

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

output== {t | t[account_number, balance]}

<u>Composition of relational operations</u>

Relational algebra operations can be composed together into relational-algebra expression. This required for complicated query.

Example: Find those customers who say in Kathmandu.

$$\Pi_{customer\_name}(\sigma_{customer\_city="Kathmandu"}(customer))$$

Union Operation

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

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

For $r \cup s$ 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 i^{th} column of r must deals with same type of domain of i^{th} column of s)

Example: Find all customers with either account or loan.

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

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

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

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

$\Pi_{customer\_name}$(depositor) - $\Pi_{customer\_name}$(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 $t \in r$, $q \in s$. Formally,

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

Example 1:

A	B
	1
β	2

Relation r

C	D	E
	10	a
β	10	a
β	20	b
	10	b

Relation s

r×s:

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

$\sigma_{A=\alpha}(r \times s)$:

A	B	C	D	E
	1		10	a
	1	β	10	a
	1	β	20	b
	1		10	b

Example 2:

customer_name	loan_number
X	L01
Y	L02

Relation borrower

loan_number	branch_name	amount
L01	B1	5000
L02	B2	6000

Relation loan

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

$\Pi_{customer\_name}(\sigma_{borrower.loan\_number=loan.loan\_number}(\sigma_{branch\_name="B1"}(borrower \times loan)))$

Process:

$borrower \times loan$

customer_name	borrower.loan_number	loan.loan_number	branch_name	amount
X	L01	L01	B1	5000
X	L01	L02	B2	6000
Y	L02	L01	B1	5000
Y	L02	L02	B2	6000

$\sigma_{branch\_name="B1"}(borrower \times loan)$

customer_name	borrower.loan_number	loan.loan_number	branch_name	amount
X	L01	L01	B1	5000
Y	L02	L01	B1	5000

$\sigma_{borrower.loan\_number=loan.loan\_number}(\sigma_{branch\_name="B1"}(borrower \times loan))$

customer_name	borrower.loan_number	loan.loan_number	branch_name	amount
X	L01	L01	B1	5000

$\Pi_{customer\_name}(\sigma_{borrower.loan\_number=loan.loan\_number}(\sigma_{branch\_name="B1"}(borrower \times loan)))$

customer_name
X

The Rename Operation

The result of relational-algebra expression does not have a name to refer it. It is better to give name to result relation. The rename operator is denoted by lower case Greek letter rho ( ). Rename operation in relation-algebra expressed as

$$_{x}(E)$$

where E is a relational algebra expression and x is name for result relation. It returns the result of expression E under the name x.

Since a relation r is itself a relational-algebra expression thus, the rename operation can also apply to rename the relation r (i.e. to get same relation under a new name). Rename operation can also used to rename attributes of relation. Assume a relational algebra expression E has arity n. Then expression

$$_{x(A1,A2,\ldots,An)}(E)$$

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

Example 1: Find the largest account balance in the bank.

$$\Pi_{balance}(account) - \Pi_{account.balance}(\sigma_{account.balance<d.balance}(account \times \ {}_{d}(account)))$$

Process:

account_number	**balance**
A1	500
A2	600
A3	700

Relation account

Account_number	balance
A1	500
A2	600
A3	700

Relation d

$$account \times \ {}_{d}(account)$$

account.account_number	**account.balance**	**d.balance**
A1	500	500
A1	500	600
A1	500	700
A2	600	500
A2	600	600
A2	600	700
A3	700	500
A3	700	600
A3	700	700

$$\Pi_{account.balance}(\sigma_{account.balance<d.balance}(account \times \ {}_{d}(account))$$

Account.balance

500
600

$$\Pi_{balance}(account) - \Pi_{account.balance}(\sigma_{account.balance<d.balance}(account \times \quad _{d}(account)))$$

balance
500
600
700

account.balance
500
600

Output:

balance
700

Example 2: Find the names of all customers who live on the same street and in the same city as smith.

$$\Pi_{customer.customer}(\sigma_{customer.customer\_street=smith\_add.street \wedge customer.customer\_city=smith\text{-}add.city}$$
$$(customer \times \quad _{smith\_add(street,city)} \ (\Pi_{customer\_street,customer\_city}(\sigma_{customer\_name=\text{"smith"}}(customer)))))$$

3.3.2 Formal Definition of Relational Algebra

Basic expressions in relational algebra are

- Relation in a database
- A constant relation
  - A constant relation is expression by listing its tuples
  - {(A01,”B1”,500)(A02, “B2”,600)

From the basic expression we can construct other expression. Let E1 and E2 be relational algebra expression, then following are also relational-algebra expression.

$E_1 \cup E_2$
$E_1 - E_2$
$E_1 \times E_2$
$\sigma_p(E_1)$, p is a predicate on attributes in E_1.
$\Pi_s(E_1)$, s is a list of some attributes in E_1
$_x(E_1)$, x is the new name for the result of E_1

3.3.3 Additional Operations

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 $r \cap s$ define as

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

In terms of fundamental operation of relational algebra it can express as

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

Example 1:

A	B
	1
🟨	🟨
β	1

A	B
🟨	🟨
β	3

Relation r relation s

A	B
	2

$r \cap s$:

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

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

The natural join operation

The natural join operation generally needs to simplify queries that required a Cartesian product. The natural join allow to combine certain selections and a Cartesian product into one operation. It is denoted by 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 schema and finally removes duplicate attributes.

Formally, let r and s are two relations on schema R and S respectively then $r \bowtie s$ is a relation on schema $R \cup s$. That is,

$$r \bowtie s = \Pi_{R \cup S}(\sigma_{r.A1=s.A1 \wedge r.A2=s.A2 \wedge \,.\,.\, \wedge r.An=s.An}(r \times s)$$

where {A1,A2, . .,An} are common attributes in R and S.

Example 1:
Let

R=(A,B,C,D)
S=(E,B,D)

Now, Result schema=(A,B,C,D,E)
$r \bowtie s$ is define as

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

Suppose r and s are two relation as follow

A	B	C	D
	1		a
β	2		a
	4	β	b
	1		a
δ	2	β	b

Relation r

B	D	E
1	a	
3	a	β
1	a	
2	b	δ
3	b	∈

relation s

$r \bowtie s$:

A	B	C	D	E
	1		a	
	1		a	
	1		a	
	1		a	
δ	2	β	b	δ

Example 2: Find the names of all customer who have a loan at the bank

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

Same query is express by fundamental operation as follow

$$\Pi_{customer\_name}(\sigma_{borrower.loan\_number=loan.loan\_number}((borrower \times loan))$$

Example 3: Find names of all branches with customer who have account in the bank and who live in Kathmandu.

$$\Pi_{customer\_name}(\sigma_{customer\_city="Kathmandu"}\ customer \bowtie account \bowtie depositor))$$

Example 4: Find all customers who have both loan and account at the bank.

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

In the set intersection form this can be express as

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

Note 1: Let r(R) and s(S) e relations without any attributes in common. That is $R \cap S = \Phi$ then

$$r \bowtie s = r \times s$$

Note 2: Theta Join

The theta join is an extension to the natural join operations that allow us to combine a selection and a Cartesian product into a single operation with predicate on attributes.

Let relation r® and s(S), and Θ be a predicate on attributes in the schema $R \cup S$ then theta

join operation $r \bowtie_{\Theta} s$ is defined as

$r \bowtie_{\Theta} s = \sigma_{\Theta}(r \times s)$

Example: Find all customers who have loan and stay in Kathmandu.

$\Pi_{customer\_name}(borrower \bowtie_{customer\_city="Kathmandu"} loan)$

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
- Pad tuples from the right relation that that did not match any from the left relation
- Add them to the result of full outer-join.

Example:

Consider relations

loan_number	branch_name	amount
L01	B1	500
L02	B2	600
L05	B1	700

Relation loan

customer_name	loan_number
X	L01
Y	L02
Z	L07

Relation borrower

Natural join (Inner join)

$\text{Loan} \bowtie \text{borrower}$

loan_number	**branch_name**	**amount**	**customer_name**
L01	B1	500	X
L02	B2	600	Y

Left outer-join

$\text{loan} ⟕ \text{borrower}$

loan_number	branch_name	amount	customer_name
L01	B1	500	X
L02	B2	600	Y
L05	B1	700	null

Right outer-join

$\text{Loan} ⟖ \text{borrower}$

loan_number	**branch_name**	**amount**	**customer_name**
L01	B1	500	X
L02	B2	600	Y
L07	null	null	Z

Full outer-join

$\text{Loan} ⟗ \text{borrower}$

loan_number	**branch_name**	**amount**	**customer_name**
L01	B1	500	X
L02	B2	600	Y
L05	B1	700	null
L07	null	null	Z