Unit 1: Introduction To Databases

1.1 Introduction:

Database Management System (DBMS):
A DBMS is a collection of interrelated data and a set of program to access those data.

Principles / Goals of DBMS:
- Safely of Introduction.
- Avoid unauthorized access of Data.
- Data recovery and concurrency.
- Data Dictionary and Maintance.
- Authentication and Authorization.
- Multiple Interface.

Applications of DBMS:
1) Banking - Customer information accounts & loan info banking transaction information.
2) Airlines - Reservation.
3) Universities - Student information & registrations and Grades.
4) Credit Card Transactions
5) Telecommunications
6) finance
7) Sales
8) Manufacturing
9) Human Resources - Employee Information salaries payroll taxes benifits.

1.2 Traditional file Based System

file based system supported by convential operating system.

- Having record of various files.
- Need Application program.

So, compare to DBMS it having some disadvantages as follows.

① Data Redandancy and Inconsistancy :
- higher storage and Access cost multiple file format, duplicate data.

② Difficulty in accessing data:

③ Data Isolation - Multiple files & format.

④ Integrity Problem - Balance < 0

⑤ Atomicity of Updates - failure may leave database in an incosistancy state.

⑥ Concurrency accepts of Multiple Users -
- It need performance.
- Two people reading a balance at same time.

⑦ Security Problem -
- Not secure
- problem of data hack &
- leak of data.

1.3 Database Approach -

The database approach is an improvement on the shared file solution as the use of database management system (DBMS) provides facilities for querying, data security and integrity and allows simultaneous access to data by a number of different users.

There are two approaches for developing any database, the top down method and bottom up method.

While these approaches appear radically different, they share the common goal of utilisizing a system by describing all of the interaction between the processes.

1.6 Advantages of Database Management System -

- Reducing Data Redundancy.
- Sharing of Data.
- Data integrity
- Data Security
- privacy
- Backup and Recovery
- Data Consistancy.

Disadvantages of DBMS -

- Database system are complex, difficult and time-consuming to design.
- Substantial hardware and software startup costs.
- Damage to database affects virtually all applications programs.
- Extensible conversion costs in moving form a file-based system to a database system.
- Initial training required for all programsers and users.

1.4 Roles in Database Environment :

1.5 History of Database Management System:

Techniques for data storage and processing have evolved over the years:

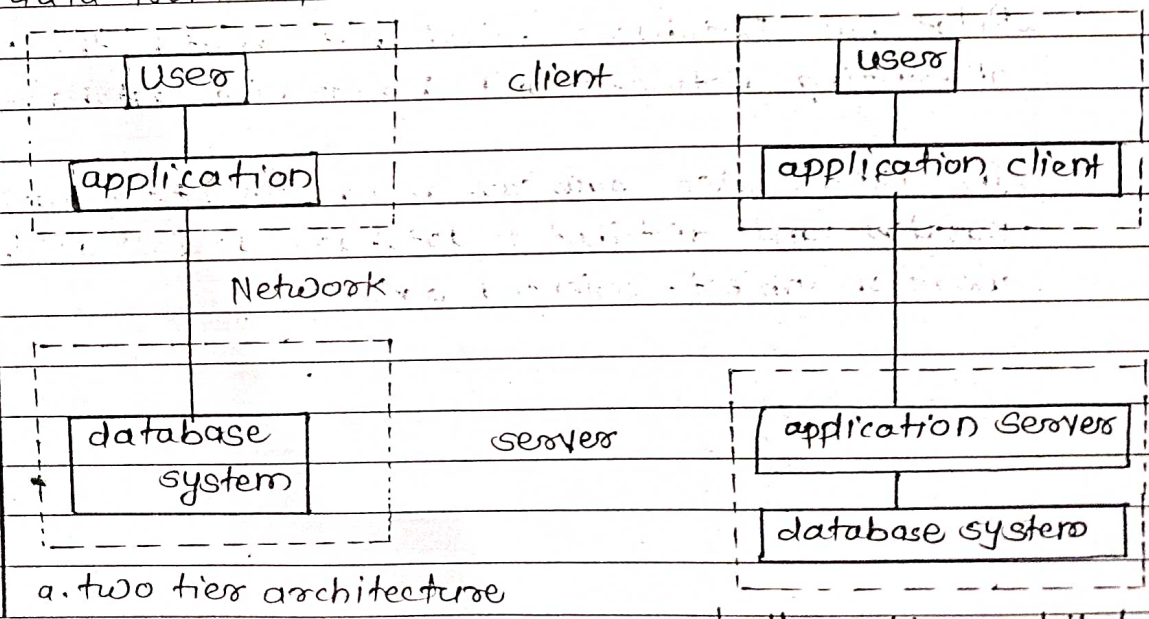
1950's and early 1960's:

Magnetic tapes were developed for data storage. Data processing tasks such as payroll were automated, with data stored on tapes.

Processing of data consisted of reading data from one or more tapes and writing data to a new tape.

Data could also be input from punched card decks, and outputs to printers.

Tapes could be read only sequentially, and data sizes were much larger than main memory; thus processing data programs were forced to process data in particular order, by reading and merging data from tapes and card decks.



Late 1960's and 1970's :

Widespread use of hard disks in the late 1960's changed the scenario for data processing greatly, since hard disks allowed direct access to data. The position of data on disk was immaterial, since any location on disk could be accessed in just tens of millisecond.

Data were thus freed from the tyranny of sequentially. With disks, network and hierarchy databases could be created that allowed data structures such as lists and trees to be stored on the disk.

Programmers could construct and manipulate those data structures.

1980's

Although academically intesting, the relational model was not used in practice initially; because of its perceived performance disadvantages;

Relational Databases could not match the performance of existing network and hierarchical databases.

In 1980's also saw much research on parallel and distributed databases as well as initial work on object-oriented databases.

Early 1990s:

The SQL language was designed primarily for decision support applications, which are query intensive, yet the mainstay of databases in the 1980s was transaction processing applications, which are update intensive.

Many database vendors introduced parallel database products in this period.

Database vendors also began to add object-relational support to their databases.

Late 1990's:

The major event was the explosive growth of the world wide web.

Databases are deployed much more extensively then even before.

Database systems how had to support very high transaction processing rates. as well as high reliability. and 624x7 availability.

Databases also had to support web interfaces to data.

1.7 Structure of Relational Databases:

A relational database consist 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.

Since a table is a collection of relationship,

Basic structure:

Account_no	branch_name	balance
A- 101	Downtown	500
A- 202	perryridge	400
A- 215	Brighton	900
A- 217	Mianus	700
A- 305	Redwood	750

fig. a

Consider the account table of fig. a. It has three column headers: account_no, branch_name, balance.

following the terminology of the relational model, we refer to these headers as attributes.

for each attribute, there is set of permitted values, called the domain of the attribute.
for branch_name attribute, for example- the domain is set of all branch names -

let, denote D_1 to set of account_no, D_2 denote set of branch_name, D_3 denote balance.

In general, account will contain only a subset of the set of all possible rows. Therefore account is subset of,

$$D_1 \times D_2 \times D_3$$

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

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

1.8 Database Schema :

We must differentiate between the database schema, which is the logical design of the database and a database instance, which is snapshot of the data in the database at a given instant in time.

The concept of a relation corresponds to the programming language notion of a variable.

The concept of relation schema corresponds to the programming-language notion of type defination.

It is convenient to give a name to a relation schema, just as we give names to type definations in programming languages.

following this notation, we use Account-schema to denote the relation schema for relation account. Thus.

Account_schema = (account_number, branch_name, balance)

We denote the fact that account is a relation on Account-schema by,

account (Account-schema)

Ingeneral, a relation schema consists of a list of attributes and their corresponding domain.

~~The concept of a relation schema consists of a list~~

The concept of a relation instance corresponds to the programming language notion of a value of a variable.

The value of a given variable may change with time, similarly the contents of a relation instance may change with time as the relation is updated. However we often simply say "relation" when we actually mean "relation instance".

As an example of a relation instance, consider the branch relation of fig. a. The schema for that relation is,

Branch_schema = (branch-name, branch-city, assets).

branch-name	branch-city	assets
Brighton	Brooklyn	710000
Downtown	Brooklyn	900000
Mianus	Horseneck	400000
North Town	Rye	300000

1.9 Keys :

Key allow us to identify a set of attributes that suffice to distinguish entities from each other.

Keys also helps uniquely relationships and thus distinguish relationships from each other.

Super key :

Super key is a set of one or more attributes that, taken collectively, allow us to identify uniquely an entity in the entity set.

for example- the customer-id attribute of the entity set customer is sufficient to distinguish one customer entity from another. Thus customer-id is a super key.

Similarly the combination of customer-name and customer-id is a superkey for the entity set customer.

The customer-name attributes of customer is not a superkey, ~~for the entity set~~ because several people might have the same name.

Candidate key :

The concept of super key is not sufficient for our purposes, since as we saw, a superkey may contain extraneous attributes.

If k is a superkey, then so is any superset of k. we are often interested in super key for which no proper subset is a super key. Such minimal superkeys are called candidate keys.

It is possible that several distinct sets of attributes could serve as a candidate key. Suppose that the combination of customer-name and customer street is sufficient to distinguish among members of the customer entity set.

Then both {customer id} and {customer name, customer street} are candidate keys.

Although the attributes customer_id and customer_name together can distinguish customer entities, their combination does not form a candidate key, since the attribute customer-id alone is candidate key.

Primary key:

We shall use the term primary key to denote a candidate key that is choasen by the database designer as the principle means of identifying entities within an entity set.

The primary should be choasen such that its attributes are never, or very rarely, changed.

1.10 Schema Diagram :-

A database schema, along with primary key and foriegn key dependencies, can be depicted pictorially by schema diagrams.

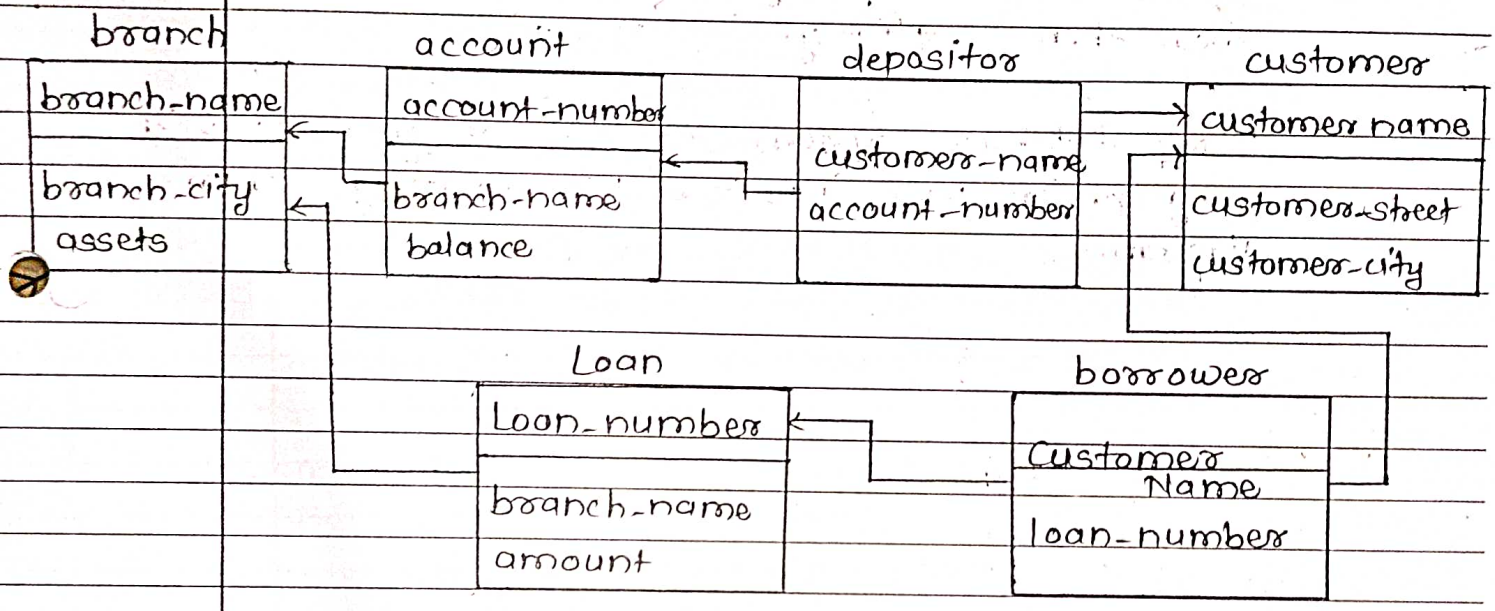


fig. Schema diagram for the banking enterprise.

above fig shows the schema diagram for our banking enterprise.

Each relation appears as a box, with the attributes listed inside it and the relation name above it.

If there are primary key attributes, a horizontal line crosses the box, with the primary key attributes listed above the line.

foriegn key dependancies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referednced relation.

Do not confuse a schema diagram with an E-R diagram.

In particular, E-R diagrams do not show foreign key attributes explicitly, where schema diagrams show them explicitly.

Many database systems provide design tools with a graphical user interface for creating schema diagrams.

1.11 Relational Query Language :

A query language is a language in which a user requests information from the database.

These languages are usually on a level higher than that of a standard programming language.

Query languages can be categorized as either procedural or non-procedural.

In a 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 desired information without giving a specific procedure for obtaining that information.

Most commercial relation-database systems offer a query language that includes elements of both the procedural and the nonprocedural approaches.

We shall study the very widely used query language SQL.

The relation algebra is procedural, whereas the tuple relational calculus and domain relational calculus are nonprocedural.

These query languages are terse and formal, lacking the "syntatic sugar" of commercial languages, but they illustrate the fundamental techniques for extracting data from the database.

Although, we shall be concerned with only queries initially, a complete data manipulation language includes not only query language, but also a language for database modification.

Such languages include commands to insert and delete tuples as well as commands to modify parts of existing tuples.

We shall examine database modification after we complete our discussion of queries.

1.12 Relational Operations:

Relational Aljebra and fundamental operations:

It is procedural query language consists of set of operations that false one or ~~more~~ two relations produce new relation in their result.

1) Select, project, union and set difference, cartesion product and rename are fundamental.

2) Set intersection, natural join, division and assignment

Select, project & rename are unary operations.

(1) Select Operations:

Select given predicate denoted by σ (sigma).

branch_name	loan_number	amount
perryridge	L-15	1500
downtown	L-16	1300
Newyorks	L-17	1200

Result of σ amount < 1600 (loan)

In general we allowing comparisions using <, >, <=, >=, ≠ in selection predicate.

We can combine the predicate by using the (and) $\wedge$ ~~or~~ & $\vee$ (or) connections.

σ branch_name = "Newyork" $\wedge$ amount ≥ 1200 (loan)

The selection predection can be operate comparision between two operations.

② The project Operation:

If we want to allow all the loan number & the amount with no fair of branch.

The project operation is a set of unary operation hence, no duplicate rows are accured.
Projection is denoted by π (pi) operation.
The arguments follows the parenthesis.

The query of loan_number & the amount can be written as,

π loan_number , amount (Loan)

Loan_number	amount
L-27	3000
L-45	800
L-18	7000
L-16	1500
L-25	900

③ Composition of Relational Operations:

If we have find the person line in Harrison we write

π customer name (σ customer-city = "Harrison" (customer))

Composing relational aljebra operations into relational aljebra expressions is just like composing arithmetic operations such as +, -, *, / into arithmetic expressions.

④ Union Operations

consider we want to find the all customer of bank either borrower or depositor the relation customer gives the relation contains all information.

Now we have given the two relation in,

cust_Name	Loan_number
Umesh	L-555
sachin	L-611
virendra	L-323
Ramesh	L-823

borrower

cust-name	acco_number
shital	A-201
sonia	A-801
swati	A 511
Manasi	A-699

Depositor

Now names of all customer having loan

Π loan_name (borrower)
Having a balance
Π cust-name (depositor)

We find the data by using binary Relation union ∪.

Π customer_name (borrower) ∪ Π customer-name (depositor).

The duplicates values are eliminated.

The relation r ∪ s is to be valid iff they ~~hold~~ hold following two conditions:

1) The relation r and s having same attribute. ie having same number of attributes.

2) The domains of ith attribute of r & ith attribute of s must be same for all i.

customer_name
Umesh
Sachin
Verendra
manasi
sonia
shital
ramesh
swati

⑤ Set difference :

denoted by $-$, we have to find the Relation is depositor but not in borrower.

The expression r-s results in the relation containing those tuples in r but not in s.

Π customer-name (depositor) $-$ Π customer-name (borrower)

customer_name
Sachin
Virendra
Ramesh

As the case with union operation. we must ensure that set difference are taken between compatible relations therefore set difference r-s is valid.

The r & s are two relations having same arity & the domain of ith attribute of r & ith attribute of s be the same.

⑥ Cartesia Product Operation

Notation X, allows combine information forms two relations Now cartesion product of Relation R_1 and R_2 be $R_1 \times R_2$.

R_1 and R_2 contain same attribute name.

Attaching the relation from which it comes.

r = borrower X loan.

As you have suspended we construct the tuple r out of each possible pair of tuples.

one form borrower and other form loan relation.

Assume n_1 tuples in borrower & n_2 tuples in loan then there are $n_1 * n_2$ ways to choosing tuple-one tuple for each relation.

Ingeneral, if we have relation $r_1(R_1)$, $r_2(R_2)$ then $r_1 \times r_2$ contains all tuples int. such that r_1 for r_1 & r_2 for r_2.

$$t[R_1] = t_1[R_1] \;\&\; t[R_2] = t_2[R_2]$$

Now we have to find customer having loan at perryridge branch should be,

$\sigma_{branch\ name = "perryridge"}(borrower \times loan)$

cust-name	loan-no	branch-name	loan-no	amount
Johes	L-17	perryridge	L-15	1500
Jones	L-17	perryridge	L-16	1300
smith	L-23	perryridge	L-15	1500
smith	L-23	—11—	L-16	1300
ityes	L-15	—11—	L-15	1500
ityes	L-15	—11—	L-16	1300
Jackson	L-14	—11—	L-15	1500

⑦ The Rename Operations :

symbol - Greek letter rho (ρ)
expression E

$\rho_x(E)$

Result of expression E under x

x = Trivial (Relational algebra expression)

give the same Relation under new name.

The rel^n algebra expression E with n arity

$\rho_x(A_1, A_2, \ldots\ldots A_n)(E)$

$A_1, A_2, \ldots\ldots A_n$ = Attribute name.

Now we have to find largest account balance in banks

Compute temperary Reln take set difference between relation π balance (account) $\cdot$ π temp reln to compute Temp. Reln we need to compute values of all account balance.

π account.balance (σ account.balance < d.balance (account $\times$ ρ_d (account))

d.rename = d.balance = renamed operation.

π balance (account) − π account.balance (σ account.balance (account $\times$ ρ_d (account)))

balance
900

formal defination of Relational Algebra :

1) Relation in the database
2) A constant relation

smaller subexpression E1 & E2 be relational algebra

1) E1 ∪ E2
2) E1 − E2
3) E1 × E2
4) σ_p (E1) ∀ P is predicates on attribute in E1.
5) π_S (E1) ∀ S is list consisting of some of the attribute E1.
6) ρ_x (E1) where x is the length new name for the result E1.

Additional Operatorssions

for lengthy expression the need of more extra operations.

① The set Intersection Operation

Symbol ∩

suppose we have to find customer both having loan and account.

Π customer_name (borrower) ∩ Π customername (depositor)

Now we have the two relation r & S then get

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

customer-name
Hayes
Jones
Smith

② The natural join Operation

The natural join is the binary operation that allow to combine certain selection & cartesion product into one operation.

denoted by join symbol ⋈

It removes the duplicate attribute.

we have two relations r(R) & s(S), The natural join r & s denoted by r ⋈ s is relation on schema R US.

$r \bowtie s = \pi_{R \cup S}(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n}(r \times s))$

Where $R \cap S = \{A_1, A_2, \ldots, A_n\}$

Eg- All customer having loan & account.

π customer-name (borrower $\bowtie$ depositor)

instead of

π customer-name (borrower) $\cap$ π customer-name (depositor)

Let $r(R)$ & $s(S)$ be relations without any attribute in common ie $R \cap S = \phi \Rightarrow \phi$ = empty set

Then

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

③ The Division Operation:

Indicated by $\div$

if $r_1 = \pi$ branch-name (σ branch-city = "Mumbai" (branch))

$r_2 = \pi$ customer-name, branch name (depositor $\bowtie$ account)

then,

$\frac{r_2}{r_1} = \pi$ customer-name, branch name (depositor $\bowtie$ account) $\div \pi$ branch-name (σ branch-city = "Mumbai" (branch))

④ The assignment Operation :

It is useful at the time of writing temperary variables, The assignment operation is denoted by ←

We could perform $r \div s$ as

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

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

$result = temp1 - temp2$

The relation used in subsequent expression the assignment should be assign any value at the Right of ← & assigned value at the left.

The tuple Relational Calculus :

In general we provide a sequence of procedures that generate the SQL query. In the other thing we generate the information at any order called Non procedural

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

ie set of tuple t is predicate P is true for t.

① Example Queries :

If we want to find to find branch name-loan, number, amount of loans over $ 1200.

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

If we want to write only loan number

$\exists t \in r (\Phi(t))$

t = tuple $\Phi(t)$ is predicate which is true

The query that shall consider next uses implication denoted by $\Rightarrow$

The formula $p \Rightarrow \Phi$ implies p implies Φ.

ie p is true then Φ must be true

$p \Rightarrow \Phi$ logically equivalent to $\neg p \vee \Phi$

② formal defination

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

$\forall$

P = formula

The tuples said to be free variables unless it is quantified by a $\exists$ or $\forall$

The atom can be bounded by:

1) $s \in r$ $\forall$ s = tuple variable

r = Relation

2) $s[x] \ominus u[y]$ $\forall$ s & u = tuple variables

x is defined by s & y is by u

$\ominus$ comparision operator like, $<, >, \leq, \geq, \neq$.

3) $s[x] \ominus c$ where s is tuple variable, x is attribute on which s is defined.

$\ominus$ = comparision operator

c = constant

Rules for building an atom:

1) An atom is a formula
2) If P_1 is formula then so are $\neg(P_1)$ & (P_1)
3) If P_1 & P_2 are formula then, $P_1 \vee P_2$, $P_1 \wedge P_2$ & $P_1 \Rightarrow P_2$.
4) if $P_1(s)$ is formula containing free tuple variables s & r is a relation then,

$\exists s \in r\,(P_1(s))$ and $\forall s \in S \in r(P_1(s))$

As the above formulas the equivalence are impor. of as follows:

1) $P_1 \wedge P_2$ is equivalant to $\neg(\neg P_1 \vee \neg P_2)$
2) $\forall t \in r\,(P_1(t))$ is equivalent to $\neg\exists t \in r\,(\neg P_1(t))$.
3) $P_1 \Rightarrow P_2$ is equivalent to $\neg P_1 \vee P_2$.

(3) Safety of Expressions

Tuple relation calculas generate infinite relation.
$\{t \mid \neg(t \in loan)\}$

It contain infinite of tuple that even not present in Database.

for avoiding this domain is used like atom(P).
P is set of values refrenced by P. it contain all inside and output values of domain

Eg - $dom(t \in loan \wedge t[amount] > 1200)$
it contain 1200 as well as values in set.

④ Expressive Power of Language:

It restricted to safe expressions is equivalent in expressive power of relational algebra it contains its own relational algebra calculations.

The domain Relational Calculus:

The 2nd form of relational calculus called domain relational calculus.

Use the domain variables to take values form attributes ~~relational calculus~~ rather than entire tuple closely related to tuple relational calculus.

① Example Queries:

find the branch name, loan number and amount for loans of over $1200.

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

② formal Defination:

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

$x_1, x_2 \ldots\ldots x_n$ = domain variables

P = formula composed of atoms

it contain one of the following form.

1) $\langle x_1, x_2, \ldots\ldots x_n \rangle \in r$, $\forall$ r = reln on n attributes and $x_1, x_2 \ldots\ldots x_n$ are domain variables or domain constants.

2) $x \ominus y$ (domain variable) x & y & the value of $\ominus$ any related relational Algebric function $<, >, \leq, \geq, =, \neq$

3) $x \ominus c$

x = domain variable

$\ominus$ = comparision operator

c = constant

for building atom the result rules are as following.

1) An atom is formula.

2) If p is formula then $\neg P_1$ & (P_1)

3) If P_1 & P_2 are formulae then $P_1 \vee P_2$, $P_1 \wedge P_2$ $P_1 \Rightarrow P_2$.

4) If $P_1(x)$ is a formulae in x where x is a domain variable then.

$\exists x(P_1(x))$ & $\forall x(P_1(x))$ are also formulae

As a Notational shorthand we write

$\exists a,b,c(P(a,b,c))$ for

$\exists a(\exists b(\exists c(D(a,b,c))))$

(3) Safely of Expressions

The safely of domain Reln & tuple reln are identical.

eg- $\{\langle b,l,a\rangle \mid \neg(\langle i,b,l,a\rangle \in loan)\}$

In tuple Relational calculus exterphially qualified variables but it not in domain relational calculus.

We say

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

It is safe if all the following are hold.

1) All values ~~from~~ ~~dom(P)~~. that appear in tuples of the expression are values form dom(P).

2) for every "there exists" subformula of form $\exists x (P1(x))$ it is true iff values x in dom $(P1)$ such that $P_1(x)$ is true

3) for every "for all" subformula of the form $\forall x (P1(x))$ the subformula is true iff $P_1(x)$ is true for all values x form dom $(P1)$.

④ Expressive power of Language's

The expressive power is more than tuple relational calculas as,

1) Relational algebra
2) Tuple relational calculus restricted to the safe expressions.
3) The domain relational calculas restricted to safe expressions.

Extended Relational Algebra Operations:

In the critical cases the relational algebra operations are not sufficient of that time.

① Generalized Projections:

They are extended by projection operation allowing arithmetic operations.

$\pi_{f_1, f_2, \ldots, f_n}(E)$.

E= Relational algebra expression.

$f_1, f_2, \ldots f_n$ = Arithmethic expression may be simple an attributes or constants.

customer Name	limit	credit balance
Umesh	6000	700
Aditya	2000	800
Arjun	4000	4000
Arvind	8000	300

The credit info^n Relation

To perform π customer_name, limit - credit_balance (credit_info)

customer_name	limit - credit balance
Umesh	5300
Aditya	1200
Arjun	0
Arvind	7700

After perform operation

② Outer Join :

Extension of join operation & deals with missing information.

customer (name, street, city)
work (name, branch-name, salary)

name	street	city	
Umesh	main	Kolhapur	← customer relation
Aditya	south	Ajara	
Arjun	west	Sangali	
Arvind	sai	Pune	
kiran	western	Asbta	

name	branch_name	salary
Umesh	CSE	50000
Aditya	Mech	40000
Arjun	IT	60000
Arvind	EE	80000
Vivek	ET	60000

← work relation

The natural join operation should be ⋈
customer ⋈ work

In the customer Reln ~~is~~ kiran is present but not in work & vivek in works not in customer

The natural join ⋈ operation will gave

Name	street	city	branch_name	salary
Umesh	main	kolhapur	CSE	50000
Aditya	south	Ajara	MEch	40000
Arjun	west	sangli	IT	60000
Arvind	sai	Pune	EE	9000

We use the outer join operation for avoid the loss of information.

Three forms

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

Ø We

① Left outer join ⟕

name	street	city	branch_name	salary
Umesh	main	Kolhapur	CSE	50000
Aditya	south	Ajara	Mech	40000
Arjun	west	Sangli	IT	60000
Arvind	sai	Pune	EE	90000
kiran	western	Ashta	null	null

Result of customer ⟕ work

② Right outer join ⟖

Name	street	city	branch Name	salary
Umesh	main	Kolhapur	CSE	50000
Aditya	south	Ajara	Mech	40000
Arjun	west	Sangli	IT	60000
Arvind	sai	Pune	EE	90000
vivek	~~western~~ Null	~~Ashta~~ Null	ET	60000

Now the combination of both given the results

name	street	city	branch Name	salary
Umesh	Main	Kolhapur	CSE	50000
Aditya	south	Ajara	IT	60000
Arjun	west	Sangli	Mech.	40000
Arvind	sai	Pune	EE	90000
kiran	western	Ashta	null	null
Vivek	null	null	ET	60000

Result of customer ⟗ works

③ Aggregate function

collection of values & written as a single valued Result.

Eg: the aggregate function same take the values of aggregate & written the result in sum

Eg - <1,1,3,4,4,11>

Ans. Sum = 24

Avg = Average the values 4

count = No. of values ie 6

min = min num = 1

max = max num = 11

The collection on which multivalued occurances are in order the values are occured are not relevent such collection called multisets.

Multisets contains only one copy of each element.