Traditional database: Most of the information that is stored and accessed is either textual & / or numeric.
Multimedia database: Store images, audio clips & video streams digitally.
Geographic Information Systems: can store and analyze maps, weather data & satellite images.
Data warehouses and Online Analytical Processing (OLAP): used by many companies to extract & analyze useful info. from very large databases to support decision making.
Real time & active database technology: used to control industrial & manufacturing processes.

Introduction:

Database: A Collection of related data with an implicit meaning.

Implicit properties of a database:

1) It represents some aspect of a real world, sometimes called miniworld or universe of discourse (UoD). Changes in miniworld are reflected in the database.
2) It is a logically coherent collection of data with some inherent meaning.
3) It is designed, built & populated with data for a specific purpose.

DBMS is a collection of programs that enables users to create & maintain a database.

- It is a general-purpose s/w system that facilitates the processes of defining, constructing, manipulating & sharing databases among various users & appl^n.

Defining a database involves specifying the data types, structures and constraints of the data to be stored in the database. The database definition or descriptive info. is also stored in the database in the form of a database catalog or dictionary; it is called meta-data

Constructing a database is a process of storing the data on some storage medium that is controlled by the DBMS.

Manipulating a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, & generating reports from the data.

Sharing a database allows multiple users & programs to access the database simultaneously.

- An appl^n program accesses the database by sending queries or requests for data to the DBMS.
- A query causes some data to be retrieved
- A transaction may cause some data to be read & some data to be written into the database.

Protection of a database includes system protection against H/w & S/w malfunction (or crashes) & security protection against unauthorized or malicious access.

Also DBMS must be able to maintain the database for system by allowing the system to evolve as requirements change over time.

Explain all the above concepts with an example of University database.

Characteristics of the database approach:

1. Self describing nature of a database system.
2. Insulation bet^n prog. & data & data abstraction
3. Support of multiple views of the data.
4. Sharing of data & multiuser transaction processing.

→ The database system contains not only the database itself but also a complete definition or description of database structure & constraints.

- Program-data independence } data abstraction
- Program-operation independence
- Conceptual representation of data do not include many of the details of how the data is stored or how the operations are implemented.
- Data model is a type of data abstraction that is used to provide conceptual representation.

- Support of multiple views of the data
A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored.
- Sharing of data & multiuser transaction Processing
DBMS must include concurrency control to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.
Ex: OLPT appl^n.

Transaction is an executing program or process that includes one or more database accesses, such as reading or updating of database records.
Isolation property ensures that each transaction appears to execute in isolation from other transactions, even though hundreds of tranactions may be executing concurrently.
Atomicity property ensures that either all the database operations in a transaction are executed or none are.

Actors on the scene

- DBA
- database designers
- End users
  - Casual end users
  - Naive or parametric end users
  - Sophisticated end users
  - Standalone users.
- System analyst & appn programmers (S/w Engg)

Workers behind the scene:

- DBMS system designers & implementers
- Tool developers
- Operators & maintenance personnel

Advantages of using DBMS approach

- Controlling redundancy
- Restricting unauthorized access
- Providing persistent storage for program objects
  - " storage structures for efficient query processing
  - " Backup & recovery
  - " multiple user interfaces
- Representing complex relationships among data
- Enforcing integrity constraints
- Permitting inferencing & actions using rules
- Reduced appn development time
- Flexibility, availability of up to date information
- Economies of scale.

A Brief history of database application

- Early database appl^n using hierarchical & N/w systems.
- Providing appl^n flexibility with relational databases.
- OO appl^n & the need for more complex databases.
- Interchanging data on the web for E-commerce.
- Extending database capabilities for new appl^n
- Databases vs information retrieval.

When not to use a DBMS.

Overhead cost of using DBMS are due to:

- High initial investment in H/w, S/w & training
- The generality that a DBMS provides for defining & processing data
- Overhead for providing security, concurrency control, recovery & integrity functions.

∴ use regular files under following circumstances:

- Simple, well defined database appl^n that are not expected to change
- Stringent, real time requirements for some programs that may not be met ∵ of DBMS overhead.
- No multiple-user access to data.

In a basic client/server DBMS architecture, the system functionality is distributed bet" 2 types of modules.

Client module — Server module.

Client module - typically designed so that it will run on a user workstation or personal computer. Hence, the client module handles user interaction & provides the user-friendly interfaces such as forms- or menu-based GUI's.

Server module - typically handles data storage, access, search & other functions.

Data models, schemas & instances:

Data abstraction - refers to the suppression of details of data organization & storage & the highlighting of the essential features for an improved understanding of data. One of the main characteristics of the database approach is to support data abstraction so that different users may perceive data at their preferred level of detail.

Data model is a collection of concepts that can be used to describe the structure of a database - provides the necessary means to achieve this abstraction.

Categories of data models
- High level or conceptual data model
- representational or implementation "
- Low level or physical data model.

High level data models – provide concepts that are close to the way many users perceive data. Use concepts such as entities, attributes & relationships.

Entities – represent a real world object or concepts.

Attribute – " some property of interest that further describes an entity.

Relationship – among 2 entities represents an association among 2 or more entities.

Low level data model – provide concepts that describe the details of how data is stored in the computer.

- concepts provided are generally meant for computer specialists, not for typical end users.
- describe how data is stored as files in the computer by representing info. such as record formats, record orderings & access paths. An access path is a structure that makes the search for particular database records efficient. An Index is an ex. of access path that allows direct access to data using an index term.

Representation or implementation data model – is in bet^n above 2 data models. Provide concepts that may be understood by end users but that are not too far removed from the way data is organized within the computer. This model hide some details of data storage but can be implemented on a computer system directly. Also called as, Legacy data model. Ex: N/w models, Hierarchical models.

Object data model group (ODMG) is a new family of higher level implementation data models that are closer to conceptual data models.

Schemas, Instances & database State:

The description of a database is called the database schema, which is specified during database design & is not expected to change frequently.

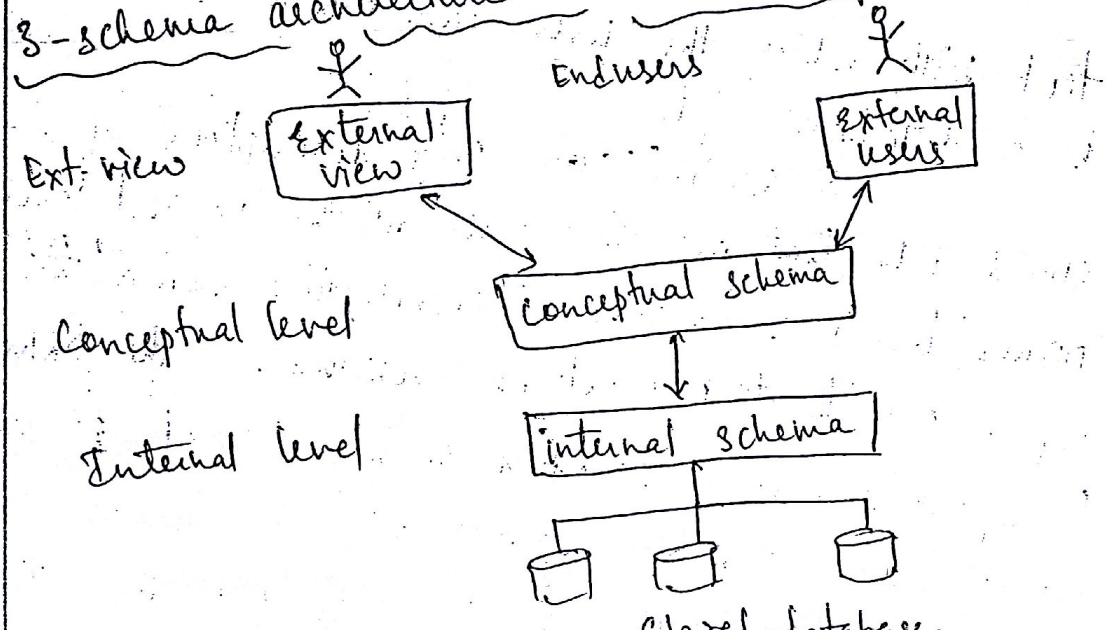
A displayed schema is called schema diagram.

Schema construct displays the structure of each record type but not the actual instances of records. (Table name)

The data in the database at a particular moment in time is called a database state or snapshot. It is also called the current set of occurrences or instances in the database.

The schema is so also called as intension & a database state is called an extension of the schema.

3-schema architecture & data Independence:



The goal of the 3-schema architecture is to separate the user applications & the physical database.

The internal level has an internal schema uses a physical data model & describes the complete details of data storage & access paths for the database.

The conceptual level has a conceptual schema which describes the structure of the whole database for a community of users. It hides the details of physical storage structures & concentrates on describing entities, data type, relationships, user operations & constraints.

The external or view level includes the no. of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in & hides the rest of the database from that user group.

The 3-schema architecture is a convenient tool with which the user can visualize the schema levels in a database system.

3-schema are only descriptions of data, the stored data that actually exists at the physical level. In a DBMS based on 3-schema architecture, each user group refers only to its own external schema... the DBMS must transform a request specified on an external schema into a request against the conceptual schema & then into a request on the internal schema for processing over the stored database.

If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view. The processes of transforming requests & results bet" levels are called mappings. These mappings may be time consuming.

Data Independence — Logical, Physical

Logical data independence is the capacity to change the conceptual schema without having to change external schemas or application programs. One may change the conceptual schema to expand the database, to change constraints or to reduce the database. Changes to constraints can be applied to the conceptual schema, without affecting the external schemas or appl" programs.

Physical data independence is the capacity to change the internal schema without having to change the conceptual schema. ∴ the external schemas need not be changed as well.

Generally, physical data independence exists in most databases & file environments in which the exact location of data on disk, H/w details of storage encoding, placement, compression, splitting, merging of records ... are hidden from the user. Applications remains unaware of these details. On the other hand, logical data independence

is very hard to come by, because it allows structural & constraint changes without affecting appl" programs - a much stricter requirement.

Database Languages :- The DBMS must provide appropriate languages & interfaces for each category of users.

Data Definition Language (DDL) is used by DBA & by database designers to define conceptual & internal schemas. The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs & to store the schema description in the DBMS catalog.

Storage Definition Language (SDL) is used to specify the internal schema. The mapping bet" the 2 schemas may be specified in either one of these languages.

View definition Languages (VDL) is used to specify user views & their mappings to the conceptual schema, but in most DBMS's the DDL is used to define both conceptual & external schemas.

Once the database schemas are compiled & the database is populated with data, users must have some means to manipulate the database; known as Data Manipulation Language :- Ex: Retrieval, Insertion, Deletion & modification of the data.

DML
- High level or nonprocedural DML
- Low level or procedural DML.

High level DML can be used on its own to specify complex database operations concisely. Also known as set-at-a-time DML.

Low level DML must be embedded in a general purpose programming language. Also called as record-at-a-time DML.

A query in high level DML often specifies which data to retrieve rather than how to retrieve it. ∴ such languages are also known as declarative.

Whenever DML commands, whether high or low level, are embedded in a general purpose programming language, that language is called host language & the DML is called data sublanguage. On the other hand, a high level DML used in a standalone interactive manner is called a query language. In general, both retrieval and update commands of a high level DML may be used interactively & are hence considered part of the query language.

DBMS Interfaces
- Menu based interfaces for web based clients or browsing
- Form-based interfaces
- GUI
- Natural language interfaces
- Speech I/P & O/P.
- Interfaces for parametric users
- Interfaces for the DBA.

(14)

The database System Environment

Fig 2-3 Pageno 44

Explanation w.r.t. fig.

Database System Utilities: To help DBA to manage database system
- Loading
- Backup
- Database storage reorganization
- Performance monitoring

Tools, application Environments & comm^n facilities:

Data dictionary or data repository system is another tool available to database designers, users. DD stores other info such as design decisions, usage standards, appl^n program descriptions & user information. Such a system is also called as information repository. A DD utility is similar to DBMS catalog but it includes a wider variety of info & is accessed mainly by users rather than by the DBMS s/w.

Appl^n development environments: Ex: powerbuilder (sybase), J builder etc.

Comm^n s/w - whose function is to allow users to locations remote from the database system site to access the database through terminals, workstations or local PC.

Centralized & client/server architectures for DBMS's

Basic client/server architecture was developed to deal with computing environments in which a large no. of PC's, workstations, file servers, printers, database servers, web servers & other equipment are connected via a n/w. The **client machines** provide the user with the appropriate interfaces to utilize these servers, as well as with local processing power to run local applications. This concept can be carried over to s/w, with specialized programs - such as a DBMS or a CAD package - being stored on specific server m/c's & being made accessible to multiple clients.

Fig 2.4, 2.5, 2.6 / Page No. 45, 46.

The concept of client/server architecture assumes an underlying framework that consists of many PC's & workstations as well as a smaller no. of mainframe m/c's, connected via LAN's & other types of computer n/w's.

A **client** in this framework is typically a user m/c that provides user interface capabilities & local processing. When a client requires access to additional functionality - such as database access - that does not exist at that m/c, it connects to a server that provides the needed functionality.

A **server** is a system containing both h/w & s/w that can provide services to the client m/c's, such as file access, printing, archiving or database access.

(16)

2-tier client/server (c/s) architectures for DBMS's

The RDBMS's created a logical dividing point bet" client and server (since SQL provides a standard language for RDBMS's) Hence the query & transaction functionality related to SQL processing remained on the server side. In such architecture, the server is often called a query server or transaction server because it provides these 2 functionalities. In an RDBMS the server is also often called an SQL server.

In such c/s architecture, user interface programs & appl" programs can run on the client side. When DBMS access is reqd, the program establishes a connection to the DBMS (which is on the server side) once the connection is created, the client program can communicate with the DBMS. A standard called ODBC provides an API, which allows client-side programs to call the DBMS, as long as both client & server m/c's have the necessary s/w installed. A client program can actually connect to several RDBMS's & send query & transaction requests using the ODBC API, which are then processed at the server sites. Any query results are sent back to the client program, which can process or display the results as needed. Ex: JDBC (Java)

The second approach to c/s architecture was taken by some Object oriented DBMS's, where the s/w modules of the DBMS were divided bet" client & server in a more integrated way.

for Example, the server level may include the part of DBMS s/w responsible for handling data storage on disk pages, local concurrency control & recovery, buffering & caching of disk pages & other such functions. Meanwhile, the client level may handle the user interface, DI functions, DBMS interactions with programming language compilers, global query optimization, concurrency control & recovery across multiple servers, structuring of complex objects from the data in the buffers & other such func. In this approach, the c/s interaction is more tightly coupled & is done internally by the DBMS modules - some of which reside on the client & some on the server - rather than by the users. The exact division of functionality varies from system to system. In such a c/s architecture, the server has been called a data server because it provides data in disk pages to the client. This data can then be structured into objects for the client programs by the client-side DBMS s/w itself. The advantages of 2-tier architecture is its simplicity & seamless compatibility with existing systems. The emergence of the web changed the roles of clients & server, leading to the 3-tier architecture.

3-tier & n-tier architectures for web applications

The intermediate layer or middle tier; also called as appl^n server or web server, depending on the appl^n.

(09/6) Fig 2.7 P.No. 48

Conceptual modeling is a very important phase in designing a successful database application. Database appl" refers to a particular database & the associated programs that implement the database queries & updates. Part of the database appl" will require the design, implementation & testing of appl" programs.

ER model - Popular high level conceptual data model.

ER diagram - diagrammatic notation associated with the ER model.

UML - Universal Modeling Language → Object modeling methodology. - popular in s/w Engg & design. This methodology goes beyond database design to specify detailed design of s/w modules & their interactions using various types of diagrams. Here we have class diagrams (similar to ER diagrams) In class diagrams, operations on objects are specified, in addition to specifying the database schema structure. Operations can be used to specify the functional requirements during database design.

Using High level Conceptual data models for database design

Step 1: Requirements collection & analysis.

data requirements ⟶ Functional requirements
operations / transactions

Step 2: To create a conceptual schema for a database detailed description of entity types, relationships & constraints

Step 3: The basic data model operations can be used to specify the high level user operations identified during functional analysis.

13/1
No-5

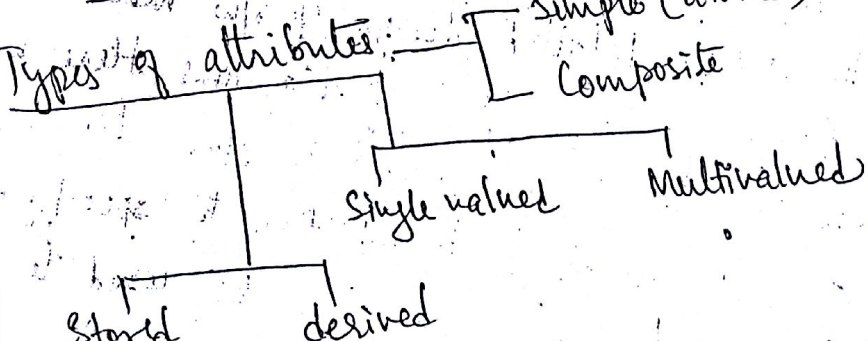
step 4: Actual implementation of the database using commercial DBMS. This step is called logical design or data model mapping.

5: Physical design phase - during which the internal storage structures, indexes, access paths & file organizations for the database files are specified.

Entity types, Entity Sets, Attributes & keys.

An Entity may be an object with a physical existence or an object with a conceptual existence.

An attribute is a property that describes an entity.



Simple attribute - attributes that are not divisible

Composite attribute - attributes that can be divided into smaller subparts, which represent more basic attributes with independent meaning.

Ex: address — st. no, Door no, Location

Single valued attribute - attribute with a single value for a particular entity. Ex: Age.

Multivalued attribute - diff no. of values for a attribute. It may have a lower & upper bounds to constrain the no. of values allowed for an individual entity.

Stored Vs Derived attribute : for Ex: for a particular entity, the value of age can be determined from the current date & the value of DOB. The age attribute is called derive attribute & is said to be derivable from DOB attribute which is a stored attribute.

Null values : can be used for not applicable or unknown values.

Complex attributes : A composite & multivalued attribute can be nested arbitrarily. We can represent arbitrary nesting by grouping components of a composite attribute bet" parantheses () & separating the components with commas, & by displaying multivalued attributes bet" braces { }. Such attributes are called complex attribute

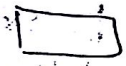
Ex: a person with more than 1 residence & each residence can have a single address & multiple phones, then person & address are themselves composite attribute.

{address_phone ({phone (area_code, phoneno)}, address (street_no, street, apartment_no), city, state, zip)) }.

Entity types, Entity Sets, keys & value sets.

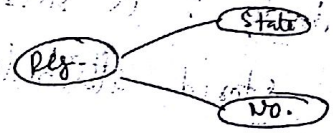
An entity type defines a collection (or set) of entities that have the same attributes.

An entity set is a collection of all entities of a particular entity type in the database at any point in time.

Entity — 

attribute — 

Multivalued attribute — 

Composite attribute are attached to their component attributes by straight lines Ex: 

An entity type describes the schema or intension for a set of entities that share the same structure.

The collection of entities of a particular entity type is grouped into an entity set, also called the extension of the entity type.

Value Sets (Domain) of attributes - typically specified using the basic data types available, like integer, string, boolean, float, enumerated type, subrange etc.

It specifies set of values that may be assigned to that attribute for each individual entity.

Mathematically, an attribute A of entity type E whose value set is V can be defined as a function from E to the power set P(V) of V:

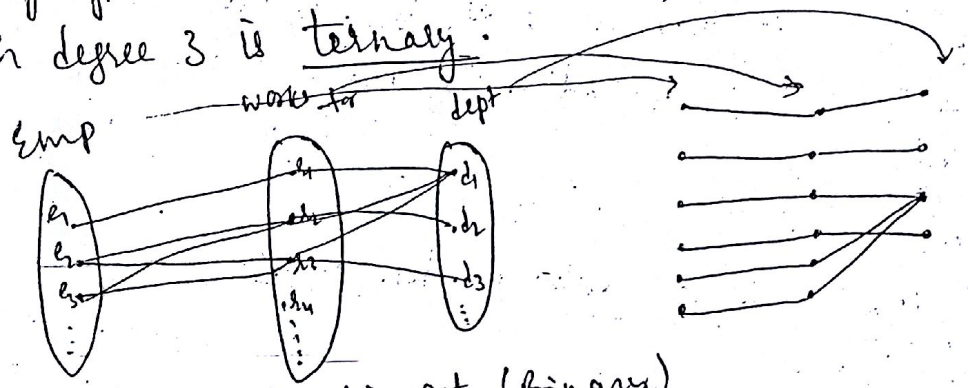
$$A : E \rightarrow P(V)$$

Relationship types, Relationship sets, Roles & Structural constraints

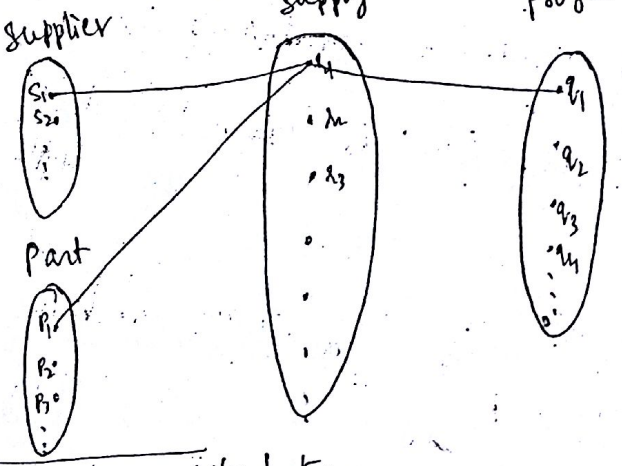
A relationship type R among n entity types $E_1, E_2 \ldots E_n$ defines a set of associations or a relationship set among from these entity types.

Relationship degree, Role names & Recursive Relationships

The degree of a relationship type is the no. of participating entity types. A relationship type of degree 2 is binary & one with degree 3 is ternary.



'works_for' → relationship set (Binary)

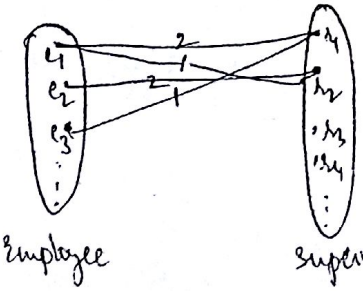


(ternary)

Relationship as Attributes

Role Name – not necessary in relationship types where all the participating entity types are distinct.

Recursive relationships – In some cases, the same entity type participates more than once in a relationship type in different roles. In such cases the role name becomes essential for distinguishing the meaning of each participation. Such relationship types are called recursive relationship.



e_1 supervises e_2
e_2 is supervised by e_3

Constraints on Relationship types — Cardinality ratio, Participation

Cardinality ratio for binary relationships specifies max. no. of relationship instances that an entity can participate in.
Ex: DEPARTMENT : EMP is of cardinality ratio 1:N (works_for)
Other cardinality ratios could be 1:1, N:1, M:N
1:1 (MANAGES), M:N (works_ON)

Participation Constraints & existence dependencies:

Total, Partial

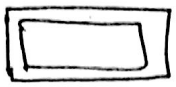
Cardinality ratios } structural constraints
Participation constraints }

In ER diagram total participation is shown by double lines ═
Partial " " " " single " —

Weak Entity types - do not have any pri key attributes on its own.
Regular entity types do have a pri key attribute are called strong entity type. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. Such other entity types are called identifying or owner entity type or parent entity type or dominant entity type.
Relationship that relates a weak entity type to its owner the identifying relationship of weak entity type or child entity type or subordinate entity type.
Weak entity type always has a total participation constraint w.r.t. its identifying relationship ∴ a weak entity cannot be identified without an owner entity.

Ex: no 2 dependents of same employee has same first name

weak entity type also called has partial key or discriminator.

↓ Represented by 

Partial key attribute is underlined with dotted line (....)

Refining the ER design for COMPANY database

- ER diag., Naming Conventions & design issues – all notations.
- Proper Naming of schema constructs
- Design choices for ER conceptual design.
- Alternate notation for ER diagrams
- UML - Universal Modeling Language

ER notation for specifying structural constraints on relationship is a pair of integer no's. (min, max) where $0 \leq min \leq max$ & $max \geq 1$. The no's mean that for each entity e in E, e must participate in at least min & at most max relationship instances in R at any point in time. In this method, min = 0 implies partial participation, whereas min > 0 implies total participation.

Usually one uses either the cardinality ratio / single-line/double line notation or (min, max) notation. (Min, max) notation is more precise as it can be used to specify structural constraints for relationship type of any degree.

P.T.O.

Enhanced Entity Relationship (EER) model :-

Subclasses
Superclasses
Inheritance
Specialization → the process of defining a set of subclasses of an entity type; this entity type is called superclass of the specialization.
Generalization
↓
Refers to the process of defining a generalized entity type from the given entity types.

(d). implies disjoint

(o). implies nondisjoint

Formal definitions for the EER model concepts

A Class is a set or collection of entities. This includes any of the EER schema constructs that group entities, such as entity types, subclasses, superclasses & categories.

A subclass S is a class whose entities must always be a subset of the entities in another class, called the superclass C of the superclass/subclass (or IS-A) relationship. We denote such a relationship by C/S. For such a superclass/subclass relationship, we must always have,

$S \subseteq C$.

A specialization $Z = \{S_1, S_2, \ldots S_n\}$ is a set of subclasses that have the same superclass G; ie. G/S_i is a superclass/subclass relationship for $i = 1, 2, \ldots n$. G is called a generalized entity type or superclass of the specialization or a generalization of the subclass $\{S_1, S_2, \ldots S_n\}$.

Z is said to be total if we always (at any point in time) have $\bigcup_{i=1}^{n} S_i = G$.

otherwise, Z is said to be partial. Z is said to be disjoint if we always have

$S_i \cap S_j = \phi$ (empty set) for $i \neq j$

otherwise Z is said to be overlapping.

(d) → disjoint } in specialization
(o) → overlap }

$\cup$ → set union operation

Total participation (specialization) constraint specifies that every entity in the superclass must be a member of at least one subclass in the specialization. (shown by double line = in EER diagram)

Partial specialization allows an entity not to belong to any of the subclass. (shown by — single line)

∴ 4 possible constraints on specialization

1) Disjoint, total
2) Disjoint, partial
3) Overlapping, total
4) Overlapping, partial

ER-to-Relational Mapping algorithm: (Fig 7.1 & Fig 7.2)

Step 1: Mapping of regular entity types

2: " " weak " "

3: " " binary 1:1 Relationship types

- Foreign key approach
- Merged relation approach
- cross reference or relationship relation approach

4: Mapping of binary 1:N Relationship types

5: " " " M:N " "
Here we must create separate "relationship relation.

6: Mapping of multivalued attribute.
For each multivalued attribute, create a new relation R

7: Mapping of N-ary relationship type
for N > 2 create a new relation S to represent R

8: Options for mapping Specialization of Generalization.

EER constructs to Relations

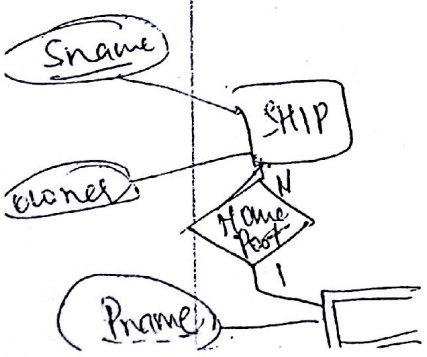
8A: Multiple relations - superclass & subclass
8B: " " subclass relations only
8C: Style relation of 1 type attribute
8D: " " with multiple type attribute

Exercise:

3.32 / 99 , 3.34 / 100 , 7.7 / 239 ,

↓

(uni, small airport)



Vaishnavi Patil

The relational model represents the database as a collection of relations. Each relation resembles a table of values or to some extent, a flat file of records.

In relational model terminology, a row is called a tuple, a column header is called an attribute & the table is called relation. The datatype describing the types of values that can appear in each column is represented by a domain of possible values.

A domain D is a set of atomic values. Atomic means, each value in the domain is indivisible as far as the relational model is concerned. A datatype or format is specified for each domain.

A relation schema R is denoted by $R(A_1, A_2, A_3 \ldots, A_n)$ where

R → Relation name (table name)

$(A_1, A_2, A_3 \ldots A_n)$ → List of its attributes

Each attribute A_i is the name of a role played by some domain D in the relation schema R. D is called domain of A_i & is denoted by $dom(A_i)$. A relation schema is used to describe a relation; R is called the name of this relation. The degree of a relation is the no. of attributes n of its relation schema.

Ex: STUDENT(Name: string, Ssn: string, Home_phone: string, Address: string, age: integer, GPa: real)

A relation or relation state r of the relation schema $R(A_1, A_2, A_3, \ldots, A_n)$, also denoted by $r(R)$, is a set of n-tuples $r = \{t_1, t_2, t_3, \ldots, t_m\}$. Each n-tuple t is an ordered list of n values $t = \langle v_1, v_2, v_3 \ldots v_n \rangle$, where each value v_i, $1 \leq i \leq n$, is an element of $dom(A_i)$.

Relation intension for the schema
Relation extension for a relation state $r(R)$ } commonly used

Thus, a relation $r(R)$ is a mathematical relation of degree n on the domains $dom(A_1), dom(A_2) \ldots dom(A_n)$, which is a subset of the cartesian product of the domains that define R:

$$r(R) \subseteq (dom(A_1) \times dom(A_2) \times \cdots \times dom(A_n))$$

The cartesian product specifies all possible combinations of values from the underlying domains.

∴ total no. of cartesian product is $|dom(A_1)| \times |dom(A_2)| \cdots \times |dom(A_n)|$

It is possible for several attributes to have the same domain. The attributes indicate different roles or interpretations, for the domain.

Characteristics of Relations

1. Ordering of tuples in a relation: A relation is defined as a set of tuples. Elements of a set have no order among them; hence tuples in a relation do not have any particular order. ie. a relation is not sensitive to the ordering of tuples.

Tuple ordering is not part of a relation definition because a relation attempts to represent facts at a logical or abstract level. But when a relation is implemented as a file or displayed as a table, a particular ordering may be specified on the records of the file or the rows of the table.

2. Ordering of values within a tuple & an alternative definition of a relation:

Alternative def^n of a relation: states, a relation schema R, $R = \{A_1, A_2, A_3 \ldots, A_n\}$ is a set of attributes, & a relation state $r(R)$ is a finite set of mappings $r = \{t_1, t_2, \ldots t_m\}$, where each tuple t_i is a mapping from R to D, & D is a union of the attribute domains, ie. $D = dom(A_1) \cup dom(A_2) \ldots dom(A_n)$

3. Values & Nulls in the tuples: Each value in a tuple is an atomic value ie. it is not divisible into components within the framework of the basic relational model. Hence, composite & multivalued attributes are not allowed. This is also known as flat relational model. This is 1NF.

∴ multivalued attributes must be represented by separate relations, & composite attributes are represented only by their simple component attributes in the basic relational model.

4. Interpretation (Meaning) of a relation :- The relation schema can be interpreted as a declaration or a type of assertion. Each tuple in a relation can be interpreted as a fact or a particular instance of the assertion. An alternative interpretation of a relation schema is as a predicate

Relational Model Notation-

- Relation schema R of degree n is denoted by $R(A_1, A_2 \ldots A_n)$
- The letters Q, R, S denote relation names.
- The letters q, r, s " " states.
- " " t, u, v " tuples.

Relational Model Constraints & Relational database schemas

In a relational database, there one will be many relations, & the tuples in those relations are usually related in various ways. The state of the whole database will correspond to the states of all its relations at a particular point in time. There are generally many restrictions or constraints on the actual values in a database state. Constraints on DB can be divided into main categories

- inherent model based or implicit constraints
- schema based or explicit constraint
  - Domain constraint
  - key constraint
  - constraints on null
  - Entity integrity constraint
  - Ref. integrity constraint
- Appl^n based or semantic constraints or business rules.
- Data Dependencies
  - functional dependencies
  - Multivalued dependencies.

Domain Constraints – specify that within each tuple, the value of each attribute A must be atomic value from the domain dom(A). The datatypes associated with domains typically include standard numeric data types for integers, real no's, characters, booleans, fixed length strings, variable length strings, date, time, time-stamp, money ..... .

STATE DOMAIN SNO_TYPE AS char(9);

Key Constraints & constraints on Null values

A relation is defined as a set of tuples. All elements of a set are distinct; ∴ all tuples in a relation must also be distinct. This means that no two tuples can have the same combination of values for all their attributes.

We denote one such subset of attributes by SK, then for any 2 distinct tuples t_1 & t_2 in a relation state r of R,

$$t_1[SK] \neq t_2[SK]$$

Any such set of attributes SK is called a superkey of the relation schema R. A superkey SK specifies a uniqueness constraint that no 2 distinct tuples in any state r of R can have the same value for SK.
A superkey can have redundant attributes.
A key K is has no redundancy
A key K of a relation schema R is a superkey of R with the additional property that removing any attribute A from K leaves a set of attributes K' that is not a superkey of R any more.
Hence a key satisfies 2 constraints:
1) 2 distinct tuples in any state of the relation cannot have identical values for (all) the attributes in the key.
2) It is a minimal superkey - ie a superkey from which we cannot remove any attributes & still have the uniqueness constraint in condition 1 hold.
Condition 1 applies to both keys and superkeys. Condition 2 is reqd. only for keys. In general, any superkey formed from a single attribute is also a key. A key with multiple attributes must require all its attributes to have the uniqueness property hold.

A relation schema may have more than one key. In this case, each of the key is called candidate key. Usually we designate one of the candidate keys as a primary key of the relation.

Relational database & Relational database schema

A relational database schema S is a set of relation schemas $S = \{R_1, R_2, \ldots R_m\}$ & a set of integrity constraints IC. A relational database state DB of S is a set of relational states $DB = \{r_1, r_2, \ldots r_m\}$ such that each r_i is a state of R_i & such that the r_i relation states the integrity constraints specified in IC.

A database state that does not obey all the integrity constraint is called an invalid state, & a state that satisfies all the constraints in IC is called a valid state.

Each RDBMS must have a DDL for defining a relational database schema.

Entity Integrity, Referential integrity & foreign keys

Entity integrity Constraint states that no primary key value can be NULL.

Ref. integrity constraint is specified bet 2 relations & is used to maintain the consistency among tuples in 2 relations.

ie the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

Ref. integrity constraints typically arise from the relationships among the entities represented by the relation schemas. This is diagrammatically shown by drawing a directed arc from each foreign key to the relation it references.

Other types of constraints

Semantic integrity constraints can be specified & enforced within the appn programs that update the database, or by using general-purpose constraint specification language. Mechanisms like TRIGGERS & ASSERTIONS can be used

Ex: CREATE ASSERTION --

Each assertion is given a constraint name & is specified via a condition similar to the WHERE clause of an SQL query.

A TRIGGER specifies an event, a condition & an action. The action is to be executed automatically if the condition is satisfied when the event occurs.

functional dependency constraint, which establishes a functional dependencies relationship among two sets of attributes X & Y. This constraint specifies that the value of X determines the value of Y in all states of a relation.

It is denoted by functional dependency $X \rightarrow Y$.

Transition constraint - defined to deal with state changes in the database.

Ex: "the sal. of an employee can only increase"

Update operations, Transactions & dealing with Constraint Violations.

INSERT - domain constraint can be violated if an attribute value is given that does not appear in the corresponding domain.

key constraint can be violated if a key value in the new tuple t already exists in another tuple in the relation r(R).

Entity constraint can be violated if the PK of the new tuple t is NULL.

Referential constraint integrity can be violated if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation.

DELETE - operation can violate only referential integrity. if the tuple being deleted is referenced by the fk from other tuples in the database.

UPDATE - may violate referential integrity.

The Transaction Concept

A database appl program running against a relational database typically runs a series of transactions. A transaction involves reading from a database as well as doing insertions, deletions & updates to existing values in database. It must leave database in a consistent state; that state must obey all the constraints we just discussed. A single transaction may involve any number of retrieval operations. A large no. of commercial appl" running against relational databases in OLTP systems are executing transactions at rates that reach several hundred/second.

Relational Algebra

2 formal languages — [Relational algebra
                       Relational Calculus.

A data model must include a set of operations to manipulate the database, in addition to the data model's concepts for defining database structure & constraints.

The basic set of operations for the relational model is called relational algebra. These operations enable a user to specify basic retrieval requests. The result of a retrieval is a new relation, which may have been formed from one or more relations. The algebra operations thus produce new relations, which can be further manipulated using operations of the same algebra. A sequence of relational algebra operations forms a relational algebra expression, whose result will also be a relation that represents the result of a database query (or retrieval request).

The relational algebra is very important for several reasons:

i) It provides a formal foundation for relational model operations.
ii) It is used as a basis for implementing & optimizing queries in RDBMSs.
iii) Some of its concepts are incorporated into the SQL for RDBMSs.

Relational algebra

- defines a set of operations for the relational model
- The operations are divided into 2 groups
  1) Set operations from mathematical set theory
  2) operations developed specifically for relational databases.
     - Ex: SELECT, PROJECT, JOIN
     - Ex: UNION, INTERSECTION, SET DIFFERENCE, CARTESIAN PRODUCT.

SELECT, PROJECT - UNARY OP.
JOIN - BINARY OPERATOR

Relational Calculus

- provides a higher level declarative notation for specifying relational queries
- an expression creates a new relation which is specified in terms of variables that range over rows of the stored database relations or over columns of the stored relations.
- There is no order of operations to specify how to retrieve the query result - specifies only what info the result should contain.
- imp because it has a firm basis in mathematical logic.

Another formal language for relational database is
2) Relational Calculus - based on mathematical calculus called Predicate calculus.

- tuple relational calculus
  - variables range over tuples
- domain relational calculus.
  - variables range over domains

UNARY Relational Operations: SELECT & PROJECT.

	Symbols
SELECT	σ
PROJECT	π

The SELECT operation is used to select a subset of the tuples from a relation that satisfies a selection condition. It can be visualized as a horizontal partition of the relation into 2 sets of tuples - those tuples that satisfy the condition & are selected, & those tuples that do not satisfy the condition & are discarded.

Ex: $\sigma_{\langle \text{selection condition} \rangle}(R)$

$\sigma_{(DNO=4 \text{ AND } sal > 25,000) \text{ OR } (DNO=5 \text{ AND } sal > 30000)}(EMP)$

The SELECT operator is UNARY ie. it is applied to a single relation. The degree of the relation resulting from a SELECT operation - its no. of attributes - is the same as the degree of R. The no. of tuples in the resulting relation is always less than or equal to the no. of tuples in R. ie. $|\sigma_C(R)| \leq |R|$ for any condition C. The fraction of tuples selected by a selection condition is referred to as the Selectivity of the condition.

SELECT is also commutative & Cascade

The PROJECT Operation: selects certain columns from the table & discards the other columns. The result of the PROJECT operation can be visualized as a vertical partition of the relation into 2 relations.

Ex: $\pi_{Lname, Fname, Salary}(EMPLOYEE)$

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

The result of PROJECT operation has only the attributes specified in <attribute list> in the same order as they appear in the list. ∴ its degree is equal to the no. of attributes in <attribute list>. The project operation removes any duplicate tuples, so the result of the PROJECT operation is a set of tuples, & hence a valid relation. This is known as duplicate elimination. Duplicate elimination involves sorting to detect duplicates & hence adds more processing. If duplicates are not eliminated, the result would be a multiset or bag of tuples rather than a set.

$\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$

* Commutativity does not hold on PROJECT.

Sequence of operations and the RENAME operation

Nesting of operations can be done.

Ex: $\Pi_{Fname, Lname, Salary}(\sigma_{Dno=5}(EMPLOYEE))$

To rename the attributes in the intermediate & result relations:

TEMP $\leftarrow \sigma_{Dno=5}(EMPLOYEE)$

R(first_name, Last_name, Salary) $\leftarrow \Pi_{Fname, Lname, Sal}(TEMP)$

for RENAME – ρ (symbol)

$\rho_{S(B_1 B_2 \cdots B_n)}(R)$ → renames both the relation & its attributes

where S – new relation name.

$B_1 B_2 \cdots B_n$ – new attribute names.

$\rho_S(R)$ → renames the relation only.

$\rho_{(B_1, B_2 \cdots B_n)}(R)$ → renames the attributes only.

Relational Algebra Operations from Set Theory

UNION, INTERSECTION & SET DIFFERENCE are Binary operations, i.e. each is applied to 2 sets (of tuples). When these tuples operations are adopted to relational database, the 2 relations on which any of these 3 operations are applied must have the same type of tuples; this condition is called union compatibility. 2 relations $R(A_1, A_2 \cdots A_n)$ & $S(B_1, B_2 \cdots B_n)$ are said to be union compatible if they have the same degree n & if $dom(A_i) = dom(B_i)$ for $1 \leq i \leq n$. This means that 2 relations have the same no. of attributes & each corresponding pair of attributes has the same domain.

We can define the 3 operations UNION, INTERSECTION & SET DIFFERENCE on 2 union-compatible relations R & S as follows:

UNION: The result of this relation, denoted by R ∪ S, is a relation that includes all tuples that are either in R or in S or in both R & S. Duplicate tuples are eliminated.

INTERSECTION: The result of this operation, denoted by R ∩ S, is a relation that includes all tuples that are in both R & S.

SET Difference (or MINUS): The result of this operation, denoted by R-S, is a relation that includes all tuples that are in R but not in S.

UNION & INTERSECTION are commutative operations. i.e $R \cup S = S \cup R$ & $R \cap S = S \cap R$.

Both UNION & INTERSECTION can be treated as n-ary operations applicable to any no. of relations ∴ both are associative operations i.e.

$R \cup (S \cup T) = (R \cup S) \cup T$ & $(R \cap S) \cap T = R \cap (S \cap T)$.

MINUS operation is not commutative. i.e $R - S \neq S - R$.

INTERSECTION can be expressed in terms of UNION & set difference as follows:

$R \cap S = R \cup S - (R - S) - (S - R)$.

The CARTESIAN PRODUCT / (CROSS PRODUCT) operation / CROSS JOIN.

→ denoted by x

→ This is also a binary set operation, but the relations on which it is applied do not have to be union compatible. In its binary form, this set operation produces a new element by combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set)

Ex: Result of $R(A_1, A_2 \ldots A_n) \times S(B_1, B_2, \ldots B_m)$ is a relation Q with $n+m$ attributes $Q(A_1, A_2, A_3 \ldots A_n, B_1, B_2 \ldots B_m)$ in that order. The resulting relation Q has one tuple for each combination of tuples - one from R & one from S.

∴ if R has n_R tuples & S has n_S tuples, R x S will have $n_R * n_S$ tuples.

The n-ary CARTESIAN PRODUCT operation is an extension of the above concept, which produces new tuples by concatenating all possible combinations of tuples from n underlying relations.

The CARTESIAN PRODUCT creates tuples with the combined attributes of 2 relations.

JOIN ⟨ EQUIJOIN
NATURAL JOIN *

BINARY Relational Operations: JOIN & DIVISION

JOIN operation:

- denoted by ⋈
- used to combine related tuples from 2 relations into single tuples.
- This allows us to process relationships among relations.
- The JOIN operation can be stated interms of a CARTESIAN PRODUCT followed by a SELECT operation.
- The result of JOIN is a relation Q with n+m attributes $Q(A_1, A_2, \ldots A_n, B_1, B_2, \ldots B_m)$ in that order. Q has one tuple for each combination of tuples – one from R & one from S – whenever the combination satisfies the join condition.
- * In JOIN, only combinations of tuples satisfying the join condition appear in the result, whereas in the CARTESIAN PRODUCT all combinations of tuples are included in the result.

A general join condition is of the form
<condition> AND <condition> AND ... AND <condition>
where each condition is of the form $A_i \theta B_j$, A_i is an attribute of R, B_j is an attribute of S, A_i & B_j have the same domain & θ is one of the comparison operators $\{=, <, >, \leq, \geq, \neq\}$. A join operation with such a general join condition is called a THETA JOIN.

Tuples whose join attributes are NULL or for which join condition is FALSE do not appear in the result. ie. the JOIN operation does not necessarily preserve all of the info in the participating relations.

Variations of JOIN :- [EQUIJOIN, NATURAL JOIN

JOIN where the only comparison operator used is =, is called EQUIJOIN.

In the result of an equijoin we always have one or more pairs of attributes that have identical values in every tuple.

NATURAL JOIN - denoted by *

used to get rid of superfluous attributes

ie. NATURAL JOIN is basically an EQUIJOIN followed by removal of the superfluous attributes.

If no combination of tuples satisfies the join condition, the result of a JOIN is an empty relation with zero tuples.

ie. if R has n_R tuples & S has n_S tuples, the result of JOIN operation $R \bowtie \langle \text{join conditions} \rangle S$ will have between zero & $n_R * n_S$ tuples. The expected size of the join result divided by the maximum $n_R * n_S$ leads to a ratio called join ~~or~~ selectivity - which is the property of each join condition. If there is no join condition, all combinations of tuples qualify & the JOIN degenates into a CARTESIAN PRODUCT or CROSS PRODUCT or CROSS JOIN.

JOIN operation is used to combine data from multiple relations so that related info can be presented in a single table. These operations are are also known as INNER JOINS.

An INNER JOIN is a type of match & merge operation defined formally as a combination of CARTESIAN PRODUCT & SELECTION.

A complete set of Relational algebra Operations

$\{\sigma, \pi, \cup, -, \times\}$ is a complete set. ie any of the other original relational algebra operations can be expressed as a sequence of operations from this set.

Ex: INTERSECTION can be expressed by using UNION & MINUS.

$$R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$$

JOIN can be specified as a CARTESIAN PRODUCT followed by SELECT operation

$$R \bowtie_{(condition)} S \equiv \sigma_{(condition)}(R \times S)$$

NATURAL JOIN can be specified as a CARTESIAN PRODUCT preceded by RENAME & followed by SELECT & PROJECT operations.

Hence various JOIN operations are not strictly necessary for expressive power of relational algebra but are used because they are convenient to use & are very commonly applied in database applications.

The DIVISION Operation ÷

The DIVISION operation is applied to 2 relations $R(Z) \div S(X)$ where $X \subseteq Z$. Let $Y = Z - X$ (& hence $Z = X \cup Y$) i.e. let Y be the set of attributes of R that are not attributes of S. The result of DIVISION is a relation $T(Y)$ that includes a tuple t if tuples t_R appear in R with $t_R[Y] = t$, & with $t_R[X] = t_S$ for every tuple t_S in S. This means, for a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with every tuple in S.

The DIVISION operation can be expressed as a sequence of π, $\times$ & $-$ operations. Most RDBMS implementations with SQL as the primary query language do not directly implement division

Notation for Query trees: A notation typically used in relational systems to represent queries internally is called query tree or query evaluation tree or query execution tree. It includes the relational algebra operations being executed & is used as a possible data structure for the internal representation of the query in an RDBMS.

A query tree represents the input relations of the query as leaf nodes of the tree, & represents the relational algebra operations as internal nodes.

An execution of the query tree consists of executing an internal node operation whenever its operands are available & then replacing the internal node by the relation that results from executing the operation. The execution terminates when the root node is executed & produces the result relation for the query.

Additional Relational Operations.

Generalized Projection:

$$\Pi_{F_1, F_2 \ldots F_n}(R)$$

where $F_1, F_2 \ldots F_n$ are functions over the attributes in relation R & may involve constants.

Aggregate functions and grouping:

common functions applied to collections of numeric values include SUM, AVERAGE, MAXIMUM, MINIMUM. COUNT is used to count the tuples or values.

- Symbol for aggregate function operation — $\mathcal{F}$ (Pronounced as script F)

Ex: $\langle \text{grouping attributes} \rangle \mathcal{F}_{\langle \text{function list} \rangle}(R)$

where $\langle$grouping attributes$\rangle$ is a list of attributes of the relation specified in R, & $\langle$function list$\rangle$ is the list of ($\langle$function$\rangle$ $\langle$attribute$\rangle$) pairs. In each such pair, $\langle$function$\rangle$ is one of the allowed functions - such as SUM, AVERAGE, MAXIMUM, MINIMUM, COUNT - and $\langle$attribute$\rangle$ is an attribute of

the relation specified by R. The resulting relation has a grouping attributes plus one attribute for each element in the function list.

Recursive Closure Operations: Recursive closure operations is applied to a recursive relationship bet" tuples of the same type. An operation called the transitive closure of relations has been proposed to compute the recursive relationship as far as the recursion proceeds.

OUTER JOIN operations:

LEFT OUTER JOIN → ⟕
RIGHT OUTER JOIN → ⟖
FULL OUTER " → ⟗

A set of operations, called outer join, can be used when we want to keep all the tuples in R, or all those in S, or all those in both relations in the result of the JOIN, regardless of whether or not they have matching tuples in the other relation.

OUTER UNION Operation: This was developed to take the union of tuples from 2 relations if the relations are not union compatible. This will take UNION of tuples in 2 relations R(X,Y) & S(X,Z) that are partially compatible, meaning that only some of their attributes, say X, are union compatible.

2 tuples t_1 in R & t_2 in S are said to match if $t_1[X] = t_2[X]$, and are considered to represent the same entity or relationship instance.

Examples of Queries in Relational Algebra:

No. 199, 200.