BIJU PATNAIK UNIVERSITY OF TECHNOLOGY, ODISHA

Lecture Notes

On

DBMS

Prepared by,

Dr. Subhendu Kumar Rath,

BPUT, Odisha.

Prepared by: Dr. Subhendu Kumar Rath

DBMS: Basic Concepts

1. Introduction
2. Disadvantages of file oriented approach
3. Database
4. Why Database
5. Database Management System(DBMS)
6. Function of DBMS
7. Advantages of DBMS and disadvantage of DBMS
8. Database Basics
9. Three level architecture of DBMS
10. Database users
11. Database language
12. Database structure

Introduction:

In computerized information system data is the basic resource of the organization. So, proper organization and management for data is required fro organization to run smoothly. Database management system deals the knowledge of how data stored and managed on a computerized information system. In any organization, it requires accurate and reliable data for better decision making, ensuring privacy of data and controlling data efficiently.
The examples include deposit and/or withdrawal from a bank,hotel,airline or railway reservation, purchase items from supermarkets in all cases, a database is accessed.

What is data:

Data is the known facts or figures that have implicit meaning. It can also be defined as it is the representation of facts ,concepts or instruction in a formal manner, which is suitable for understanding and processing. Data can be represented in alphabets(A-Z, a-z),in digits(0-9) and using special characters(+,-.#,$, etc)
e.g: 25, “ajit” etc.

Information:

Information is the processed data on which decisions and actions are based. Information can be defined as the organized and classified data to provide meaningful values.

Eg: “The age of Ravi is 25”

File:

File is a collection of related data stored in secondary memory.

File Oriented approach:

The traditional file oriented approach to information processing has for each application a separate master file and its own set of personal file. In file oriented approach the program dependent on the files and files become dependent on the files and files become dependents upon the programs

Disadvantages of file oriented approach:

1) **Data redundancy and inconsistency:**
   The same information may be written in several files. This redundancy leads to higher storage and access cost. It may lead data inconsistency that is the various copies of the same data may longer agree for example a changed customer address may be reflected in single file but not else where in the system.

2) **Difficulty in accessing data :**
   The conventional file processing system do not allow data to retrieved in a convenient and efficient manner according to user choice.

3) **Data isolation :**
   Because data are scattered in various file and files may be in different formats with new application programs to retrieve the appropriate data is difficult.

4) **Integrity Problems:**
   Developers enforce data validation in the system by adding appropriate code in the various application program. How ever when new constraints are added, it is difficult to change the programs to enforce them.

5) **Atomicity:**

   It is difficult to ensure atomicity in a file processing system when transaction failure occurs due to power failure, networking problems etc.
   (atomicity: either all operations of the transaction are reflected properly in the database or non are)

6) **Concurrent access:**
   In the file processing system it is not possible to access a same file for transaction at same time

7) **Security problems:**
   There is no security provided in file processing system to secure the data from unauthorized user access.

Database:

A database is organized collection of related data of an organization stored in formatted way which is shared by multiple users.

The main feature of data in a database are:

1. It must be well organized
2. it is related
3. It is accessible in a logical order without any difficulty
4. It is stored only once

for example:
consider the roll no, name, address of a student stored in a student file. It is collection of related data with an implicit meaning.
Data in the database may be persistent, integrated and shared.

Persistent:
If data is removed from database due to some explicit request from user to remove.
Integrated:
A database can be a collection of data from different files and when any redundancy among those files are removed from database is said to be integrated data.
Sharing Data:
The data stored in the database can be shared by multiple users simultaneously with out affecting the correctness of data.

Why Database:

In order to overcome the limitation of a file system, a new approach was required. Hence a database approach emerged. A database is a persistent collection of logically related data. The initial attempts were to provide a centralized collection of data. A database has a self describing nature. It contains not only the data sharing and integration of data of an organization in a single database.

A small database can be handled manually but for a large database and having multiple users it is difficult to maintain it, In that case a computerized database is useful.
The advantages of database system over traditional, paper based methods of record keeping are:

- **compactness**:
  No need for large amount of paper files
- **speed**:
  The machine can retrieve and modify the data more faster way then human being
- **Less drudgery**: Much of the maintenance of files by hand is eliminated
- **Accuracy:** Accurate,up-to-date information is fetched as per requirement of the user at any time.

Database Management System (DBMS):
A database management system consists of collection of related data and refers to a set of programs for defining, creation, maintenance and manipulation of a database.

Function of DBMS:

1. **Defining database schema**: it must give facility for defining the database structure also specifies access rights to authorized users.
2. **Manipulation of the database:** The dbms must have functions like insertion of record into database updation of data, deletion of data, retrieval of data
3. **Sharing of database:** The DBMS must share data items for multiple users by maintaining consistency of data.
4. **Protection of database:** It must protect the database against unauthorized users.
5. **Database recovery:** If for any reason the system fails DBMS must facilitate data base recovery.

Advantages of dbms:

Reduction of redundancies:

Centralized control of data by the DBA avoids unnecessary duplication of data and effectively reduces the total amount of data storage required avoiding duplication in the elimination of the inconsistencies that tend to be present in redundant data files.

Sharing of data:

A database allows the sharing of data under its control by any number of application programs or users.

Data Integrity:

Data integrity means that the data contained in the database is both accurate and consistent. Therefore data values being entered for storage could be checked to ensure that they fall with in a specified range and are of the correct format.

Data Security:

The DBA who has the ultimate responsibility for the data in the dbms can ensure that proper access procedures are followed including proper authentication schemas for access to the DBS and additional check before permitting access to sensitive data.

Conflict resolution:

DBA resolve the conflict on requirements of various user and applications. The DBA chooses the best file structure and access method to get optional performance for the application.

Data Independence:

Data independence is usually considered from two points of views; physically data independence and logical data independence.
Physical data Independence allows changes in the physical storage devices or organization of the files to be made without requiring changes in the conceptual view or any of the external views and hence in the application programs using the data base.
Logical data independence indicates that the conceptual schema can be changed without affecting the existing external schema or any application program.

Disadvantage of DBMS:

1. DBMS software and hardware (networking installation) cost is high
2. The processing overhead by the dbms for implementation of security, integrity and sharing of the data.
3. centralized database control
4. Setup of the database system requires more knowledge, money, skills, and time.
5. The complexity of the database may result in poor performance.

Database Basics:

Data item:
The data item is also called as field in data processing and is the smallest unit of data that has meaning to its users.
Eg: “e101”,”sumit”

Entities and attributes:

An entity is a thing or object in the real world that is distinguishable from all other objects
Eg:
Bank,employee,student
Attributes are properties are properties of an entity.
Eg:
Empcode,ename,rolno,name

Logical data and physical data :

Logical data are the data for the table created by user in primary memory.
Physical data refers to the data stored in the secondary memory.

Schema and sub-schema :

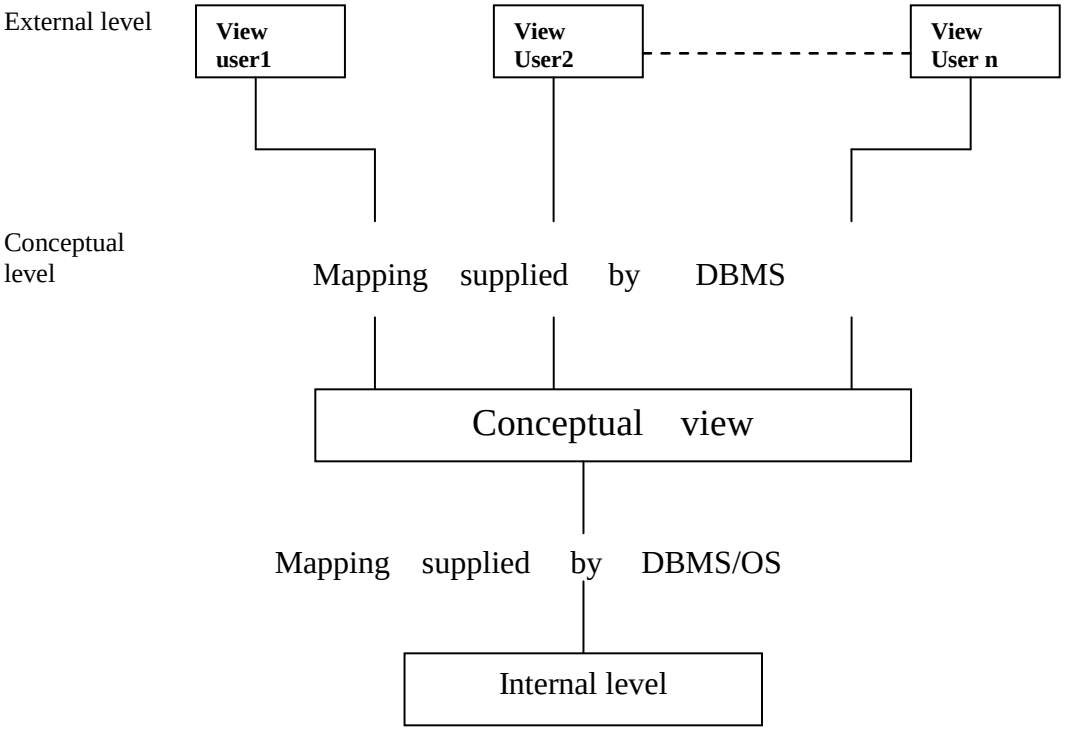
A schema is a logical data base description and is drawn as a chart of the types of data that are used . It gives the names of the entities and attributes and specify the relationships between them.

A database schema includes such information as :

- Characteristics of data items such as entities and attributes .
- Logical structures and relationships among these data items .
- Format for storage representation.
- Integrity parameters such as physical authorization and back up policies.

A *subschema* is derived schema derived from existing schema as per the user requirement. There may be more then one subschema create for a single conceptual schema.

Three level architecture of DBMS :



A database management system that provides three level of data is said to follow three-level architecture .

- External level
- Conceptual level
- Internal level

External level :

The external level is at the highest level of database abstraction . At this level, there will be many views define for different users requirement. A view will describe only a subset of the database. Any number of user views may exist for a given global or subschema.

for example , each student has different view of the time table. the view of a student of Btech (CSE) is different from the view of the student of Btech(ECE).Thus this level of abstraction is concerned with different categories of users.

Each external view is described by means of a schema called schema or schema.

Conceptual level :

At this level of database abstraction all the database entities and the relationships among them are included . One conceptual view represents the entire database . This conceptual view is defined by the conceptual schema.

The conceptual schema hides the details of physical storage structures and concentrate on describing entities , data types, relationships, user operations and constraints.

It describes all the records and relationships included in the conceptual view . There is only one conceptual schema per database . It includes feature that specify the checks to relation data consistency and integrity.

Internal level :

It is the lowest level of abstraction closest to the physical storage method used . It indicates how the data will be stored and describes the data structures and access methods to be used by the database . The internal view is expressed by internal schema.
The following aspects are considered at this level:

1. Storage allocation e.g: B-tree,hashing
2. access paths eg. specification of primary and secondary keys,indexes etc
3. Miscellaneous eg. Data compression and encryption techniques,optimization of the internal structures.

Database users :

Naive users :

Users who need not be aware of the presence of the database system or any other system supporting their usage are considered naïve users . A user of an automatic teller machine falls on this category.

Online users :

These are users who may communicate with the database directly via an online terminal or indirectly via a user interface and application program. These users are aware of the database system and also know the data manipulation language system.

Application programmers :

Professional programmers who are responsible for developing application programs or user interfaces utilized by the naïve and online user falls into this category.

Database Administration :

A person who has central control over the system is called database administrator . The function of DBA are :

1. creation and modification of conceptual Schema definition
2. Implementation of storage structure and access method.
3. schema and physical organization modifications .
4. granting of authorization for data access.
5. Integrity constraints specification.
6. Execute immediate recovery procedure in case of failures
7. ensure physical security to database

Database language :

1) Data definition language(DDL) :

DDL is used to define database objects .The conceptual schema is specified by a set of definitions expressed by this language. It also give some details about how to implement this schema in the physical devices used to store the data. This definition includes all the entity sets and their associated attributes and their relation ships. The result of DDL statements will be a set of tables that are stored in special file called data dictionary.

2) Data manipulation language(DML) :

A DML is a language that enables users to access or manipulate data stored in the database. Data manipulation involves retrieval of data from the database, insertion of new data into the database and deletion of data or modification of existing data.

There are basically two types of DML:

- **procedural**: Which requires a user to specify what data is needed and how to get it.
- **non-rocedural:** which requires a user to specify what data is needed with out specifying how to get it.

3) **Data control language(DCL):**

This language enables user to grant authorization and canceling authorization of database objects.

Elements of DBMS:

DML pre-compiler:

It converts DML statement embedded in an application program to normal procedure calls in the host language. The pre-complier must interact with the query processor in order to generate the appropriate code.

DDL compiler:

The DDL compiler converts the data definition statements into a set of tables. These tables contains information concerning the database and are in a form that can be used by other components of the dbms.

File manager:

File manager manages the allocation of space on disk storage and the data structure used to represent information stored on disk.

Database manager:

A database manager is a program module which provides the interface between the low level data stored in the database and the application programs and queries submitted to the system.

The responsibilities of database manager are:

1. **Interaction with file manager**: The data is stored on the disk using the file system which is provided by operating system. The database manager translate the the different DML statements into low-level file system commands. so The database manager is responsible for the actual storing,retrieving and updating of data in the database.
2. **Integrity enforcement:**The data values stored in the database must satisfy certain constraints(eg: the age of a person can't be less then zero).These constraints are specified by DBA. Data manager checks the constraints and if it satisfies then it stores the data in the database.
3. **Security enforcement:**Data manager checks the security measures for database from unauthorized users.
4. **Backup and recovery:**Database manager detects the failures occurs due to different causes (like disk failure, power failure,deadlock,s/w error) and restores the database to original state of the database.
5. **Concurrency control:**When several users access the same database file simultaneously, there may be possibilities of data inconsistency. It is

responsible of database manager to control the problems occurs for concurrent transactions.

query processor:

The query processor used to interpret to online user's query and convert it into an efficient series of operations in a form capable of being sent to the data manager for execution. The query processor uses the data dictionary to find the details of data file and using this information it create query plan/access plan to execute the query.

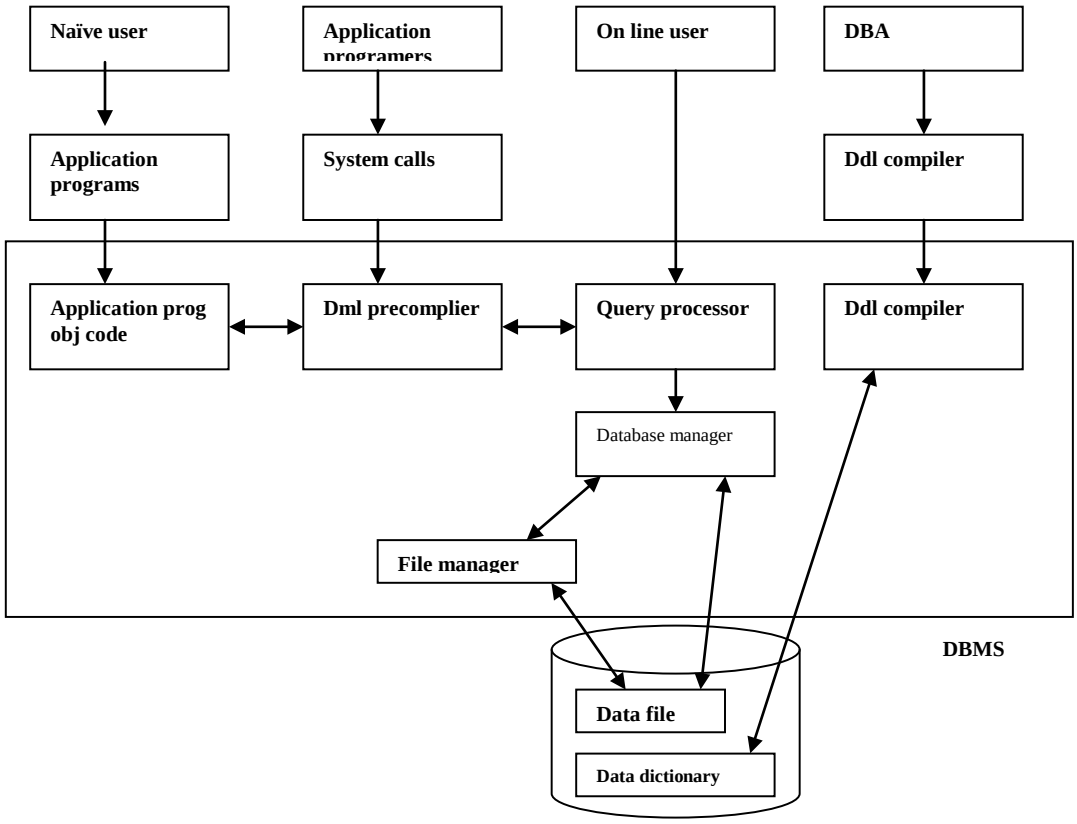
Data Dictionary:

Data dictionary is the table which contains the information about database objects. It contains information like

1. external, conceptual and internal database description
2. description of entities , attributes as well as meaning of data elements
3. synonyms, authorization and security codes
4. database authorization

The data stored in the data dictionary is called *meta data.*

DBMS STRUCTURE:



Q. List four significant differences between a file-processing system and a DBMS.

Answer: Some main differences between a database management system and a file-processing system are:

- Both systems contain a collection of data and a set of programs which access that data. A database management system coordinates both the physical and the logical

access to the data, whereas a file-processing system coordinates only the physical access.

- A database management system reduces the amount of data duplication by ensuring that a physical piece of data is available to all programs authorized to have access to it, where as data written by one program in a file-processing system may not be readable by another program.
- A database management system is designed to allow flexible access to data (i.e., queries), whereas a file-processing system is designed to allow predetermined access to data (i.e., compiled programs).
- A database management system is designed to coordinate multiple users accessing the same data at the same time. A file-processing system is usually designed to allow one or more programs to access different data files at the same time. In a file-processing system, a file can be accessed by two programs concurrently only if both programs have read-only access to the file.

Q.Explain the difference between physical and logical data independence.

Answer:

- Physical data independence is the ability to modify the physical scheme without making it necessary to rewrite application programs. Such modifications include changing from unblocked to blocked record storage, or from sequential to random access files.
- Logical data independence is the ability to modify the conceptual scheme without making it necessary to rewrite application programs. Such a modification might be adding a field to a record; an application program's view hides this change from the program.

Q. List five responsibilities of a database management system. For each responsibility, explain the problems that would arise if the responsibility were not discharged.

Answer: A general purpose database manager (DBM) has five responsibilities:

a. interaction with the file manager.
b. integrity enforcement.
c. security enforcement.
d. backup and recovery.
e. concurrency control.

If these responsibilities were not met by a given DBM (and the text points out that sometimes a responsibility is omitted by design, such as concurrency control on a single-user DBM for a micro computer) the following problems can occur, respectively:

a. No DBM can do without this, if there is no file manager interaction then nothing stored in the files can be retrieved.

b. Consistency constraints may not be satisfied, account balances could go below the minimum allowed, employees could earn too much overtime (e.g.,hours > 80) or, airline pilots may fly more hours than allowed by law.
c. Unauthorized users may access the database, or users authorized to access part of the database may be able to access parts of the database for which they lack authority. For example, a high school student could get access to national defense secret codes, or employees could find out what their supervisors earn.
d. Data could be lost permanently, rather than at least being available in a consistent state that existed prior to a failure.
e. Consistency constraints may be violated despite proper integrity enforcement in each transaction. For example, incorrect bank balances might be reflected due to simultaneous withdrawals and deposits, and so on.

Q. What are five main functions of a database administrator?

Answer: Five main functions of a database administrator are:

- To create the scheme definition
- To define the storage structure and access methods
- To modify the scheme and/or physical organization when necessary
- To grant authorization for data access
- To specify integrity constraints

Q. List six major steps that you would take in setting up a database for a particular enterprise.

Answer: Six major steps in setting up a database for a particular enterprise are:

- Define the high level requirements of the enterprise (this step generates a document known as the system requirements specification.)
- Define a model containing all appropriate types of data and data relationships.
- Define the integrity constraints on the data.
- Define the physical level.
- For each known problem to be solved on a regular basis (e.g., tasks to be carried out by clerks or Web users) define a user interface to carry out the task, and write the necessary application programs to implement the user interface.
- Create/initialize the database.

EXERCISES:

1. What is database management system
2. What are the disadvantage of file processing system

3. State advantage and disadvantage of database management system
4. What ate different types of database users
5. What is data dictionary and what are its contents
6. What are the function of DBA
7. What are the different database languages explain with example.
8. Explain the three layer architecture of DBMS.
9. Differentiate between physical data independence and logical data independence
10. Explain the function of data base manager
11. Explain meta data

CHAPTER-2

ER-MODEL

Data model:

The data model describes the structure of a database. It is a collection of conceptual tools for describing data, data relationships and consistency constraints and various types of data model such as

1. Object based logical model
2. Record based logical model
3. Physical model

Types of data model:

1. Object based logical model
   a. ER-model
   b. Functional model
   c. Object oriented model
   d. Semantic model
2. Record based logical model
   a. Hierarchical database model
   b. Network model
   c. Relational model
3. Physical model

Entity Relationship Model

The entity-relationship data model perceives the real world as consisting of basic objects, called entities and relationships among these objects. It was developed to facilitate data base design by allowing specification of an enterprise schema which represents the overall logical structure of a data base.

Main features of ER-MODEL:

- Entity relationship model is a high level conceptual model
- It allows us to describe the data involved in a real world enterprise in terms of objects and their relationships.
- It is widely used to develop an initial design of a database
- It provides a set of useful concepts that make it convenient for a developer to move from a baseid set of information to a detailed and description of information that can be easily implemented in a database system
- It describes data as a collection of entities, relationships and attributes.

Basic concepts:

The E-R data model employs three basic notions : entity sets, relationship sets and attributes.

Entity sets:

An entity is a "thing" or "object" in the real world that is distinguishable from all other objects. For example, each person in an enterprise is an entity. An entity has a set properties and the values for some set of properties may uniquely identify an entity.
BOOK is entity and its properties(calles as attributes) bookcode, booktitle, price etc .

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

Attributes:

An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set.

Customer is an entity and its attributes are **customerid, custmername, custaddress** etc.

An attribute as used in the E-R model , can be characterized by the following attribute types.

a) **Simple and composite attribute:**

simple attributes are the attributes which can't be divided into sub parts
eg: customerid,empno
composite attributes are the attributes which can be divided into subparts.
eg: name consisting of first name, middle name, last name
address consisting of city,pincode,state

b) **single-valued and multi-valued attribute:**
The attribute having unique value is single –valued attribute
eg: empno,customerid,regdno etc.
The attribute having more than one value is multi-valued attribute
eg: phone-no, dependent name, vehicle

c) **Derived Attribute:**

The values for this type of attribute can be derived from the values of existing attributes
eg: age which can be derived from (currentdate-birthdate)
experience_in_year can be calculated as (currentdate-joindate)

d) **NULL valued attribute:**
The attribute value which is unknown to user is called NULL valued attribute.

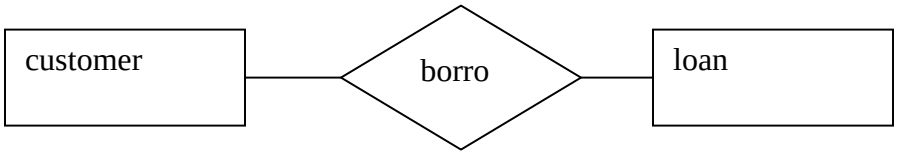
Relationship sets:

A relationship is an association among several entities.

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

{(e1,e2,…en)|e1Є E1,e2 Є E2..,en Є En}

where (e1,e2,…en) is a relation ship.



Consider the two entity sets customer and loan. We define the relationship set borrow to denote the association between customers and the bank loans that the customers have.

Mapping Cardinalities:

Mapping cardinalities or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.

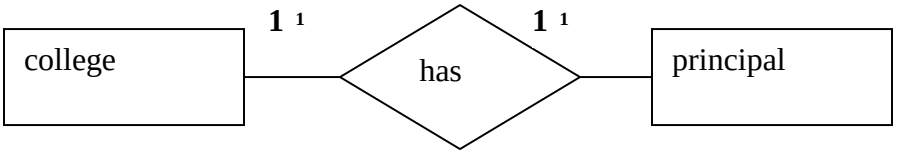
Mapping cardinalities are most useful in describing binary relationship sets, although they can contribute to the description of relationship sets that involve more than two entity sets.

For a binary relationship set R between entity sets A and B, the mapping cardinalities must be one of the following:

one to one:

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

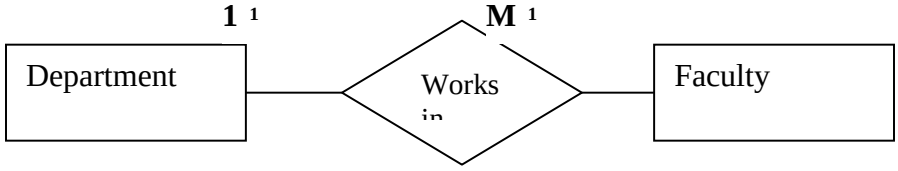
Eg: relationship between college and principal



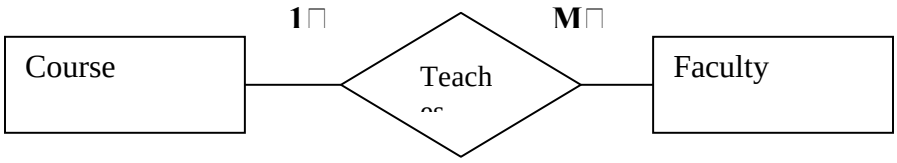
One to many:

An entity in A is associated with any number of entities in B. An entity in B is associated with at the most one entity in A.

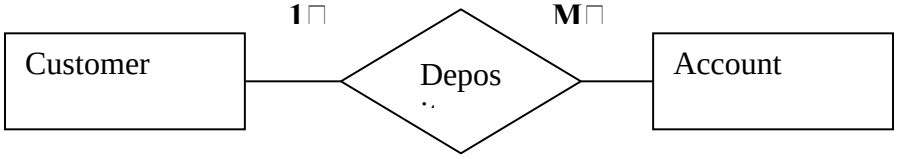
Eg: Relationship between department and faculty



Many to one:
An entity in A is associated with at most one entity in B. An entity in B is associated with any number in A.



Many –to-many:
Entities in A and B are associated with any number of entities from each other.



More about entities and Relationship:
Recursive relationships:
When the same entity type participates more than once in a relationship type in different roles, the relationship types are called recursive relationships.
Participation constraints:
The participation constraints specify whether the existence of any entity depends on its being related to another entity via the relationship. There are two types of participation constraints
Total :
.When all the entities from an entity set participate in a relationship type , is called total participation. For example, the participation of the entity set student on the relationship set must 'opts' is said to be total because every student enrolled must opt for a course.
Partial:
When it is not necessary for all the entities from an entity set to particapte ion a relationship type, it is called participation. For example, the participation of the entity set student in 'represents' is partial, since not every student in a class is a class representative.
Weak Entity:
Entity types that do not contain any key attribute, and hence can not be identified independently are called weak entity types. A weak entity can be identified by uniquely only by considering some of its attributes in conjunction with the primary key attribute of another entity, which is called the identifying owner entity.
Generally a partial key is attached to a weak entity type that is used for unique identification of weak entities related to a particular owner type. The following restrictions must hold:

- The owner entity set and the weak entity set must participate in one to may relationship set. This relationship set is called the identifying relationship set of the weak entity set.

- The weak entity set must have total participation in the identifying relationship.

Example:

Consider the entity type dependent related to employee entity, which is used to keep track of the dependents of each employee. The attributes of dependents are : name ,birthrate, sex and relationship. Each employee entity set is said to its own the dependent entities that are related to it. How ever, not that the 'dependent' entity does not exist of its own., it is dependent on the employee entity. In other words we can say that in case an employee leaves the organization all dependents related to without the entity 'employee'. Thus it is a weak entity.

Keys:

Super key:

A 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 , customer-id,(cname,customer-id),(cname,telno)

Candidate key:

In a relation R, a candidate key for R is a subset of the set of attributes of R, which have the following properties:

- *Uniqueness:* no two distinct tuples in R have the same values for the candidate key
- *Irreducible:* No proper subset of the candidate key has the uniqueness property that is the candidate key.

Eg: (cname,telno)

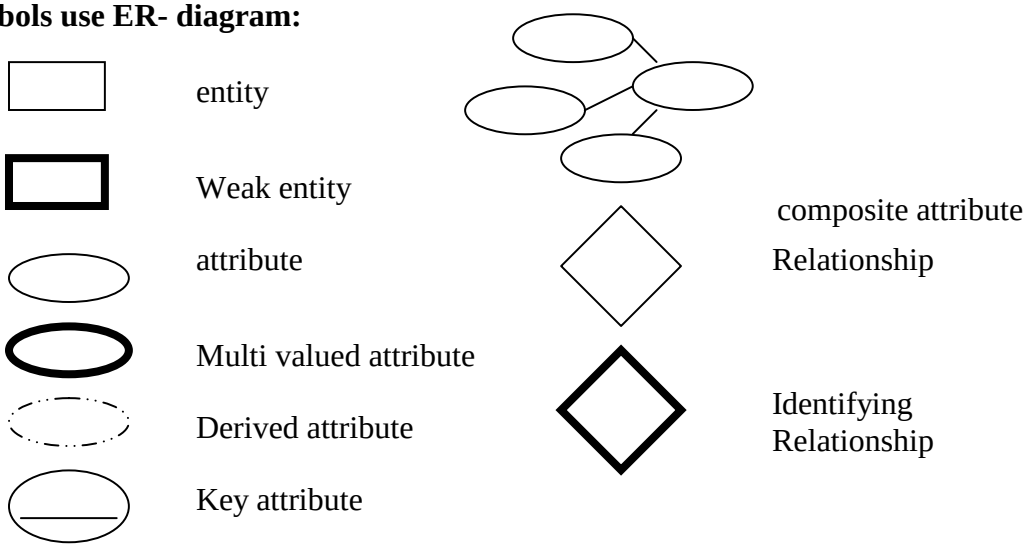
Primary key:

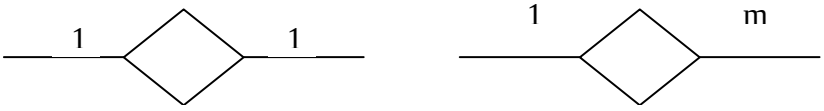
The primary key is the candidate key that is chosen by the database designer as the principal means of identifying entities with in an entity set. The remaining candidate keys if any, are called *alternate key*.

ER-DIAGRAM:

The overall logical structure of a database using ER-model graphically with the help of an ER-diagram.

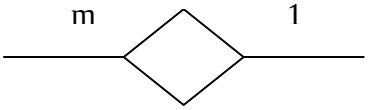
Symbols use ER- diagram:



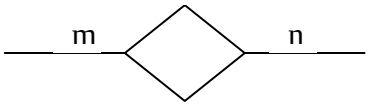


One-to -one

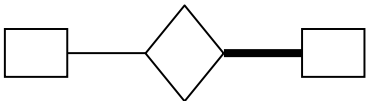
One-to -many



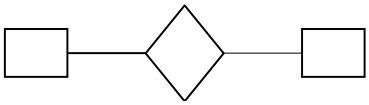
many-to -one



many-to -many



Total participation



Partial participation

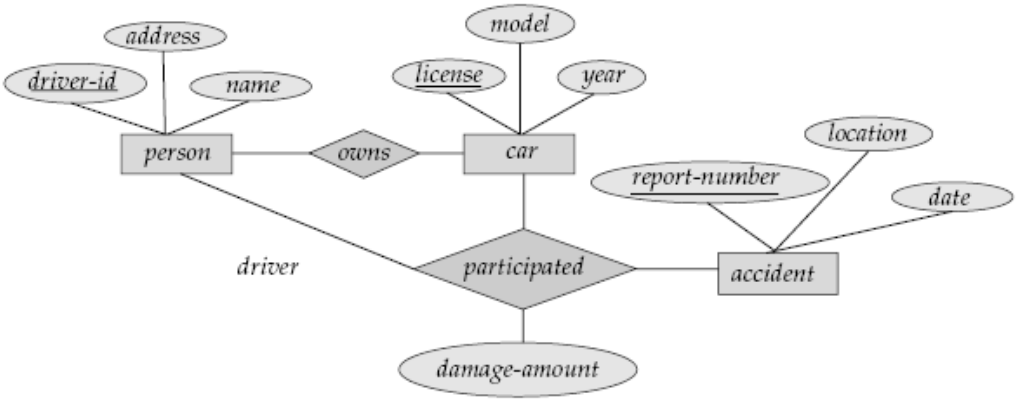


Figure 2.1 E-R diagram for a Car-insurance company.

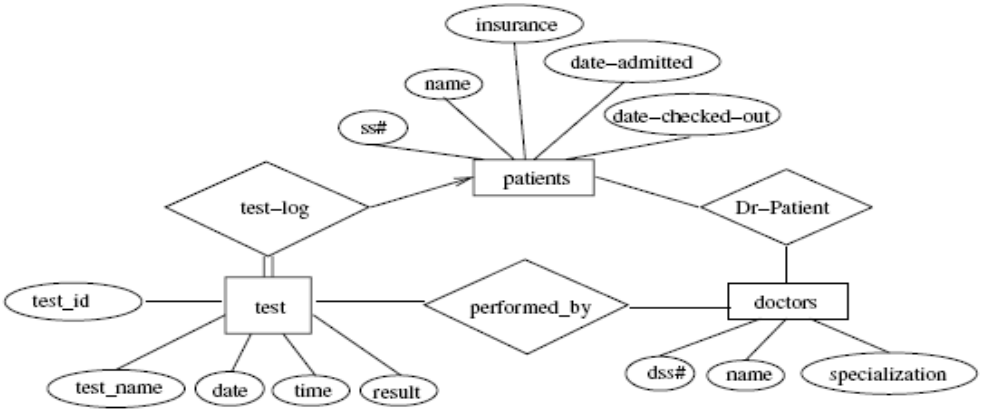


Figure 2.2 E-R diagram for a hospital.

A university registrar's office maintains data about the following entities: (a) courses, including number, title, credits, syllabus, and prerequisites; (b) course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom; (c) students, including student-id, name, and program; and (d) instructors, including identification number, name, department, and title. Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled.

Construct an E-R diagram for the registrar's office. Document all assumptions that you make about the mapping constraints.

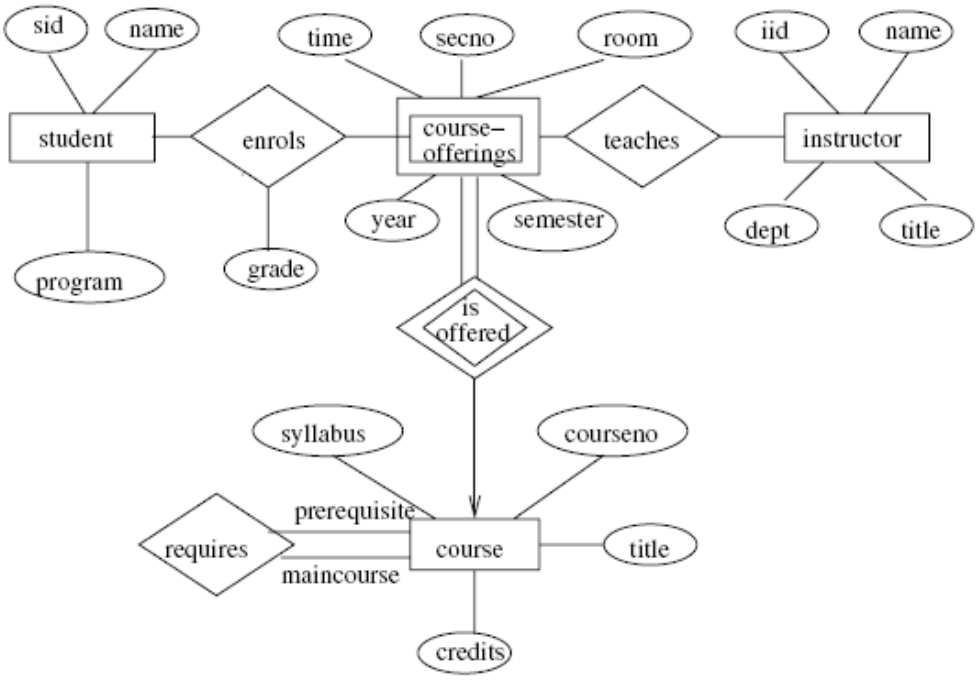


Figure 2.3 E-R diagram for a university.

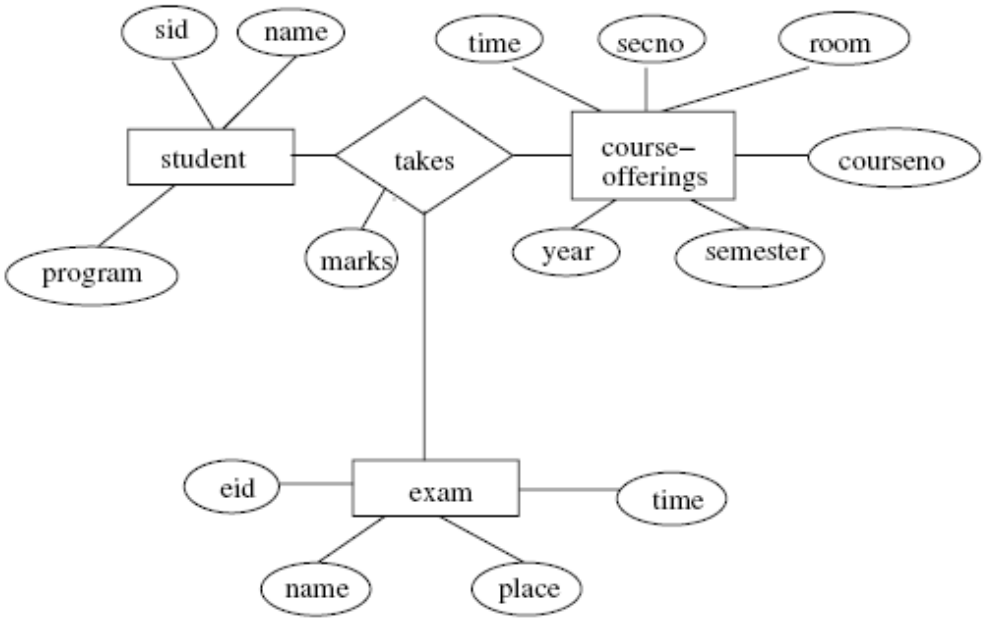


Figure 2.4 E-R diagram for marks database.

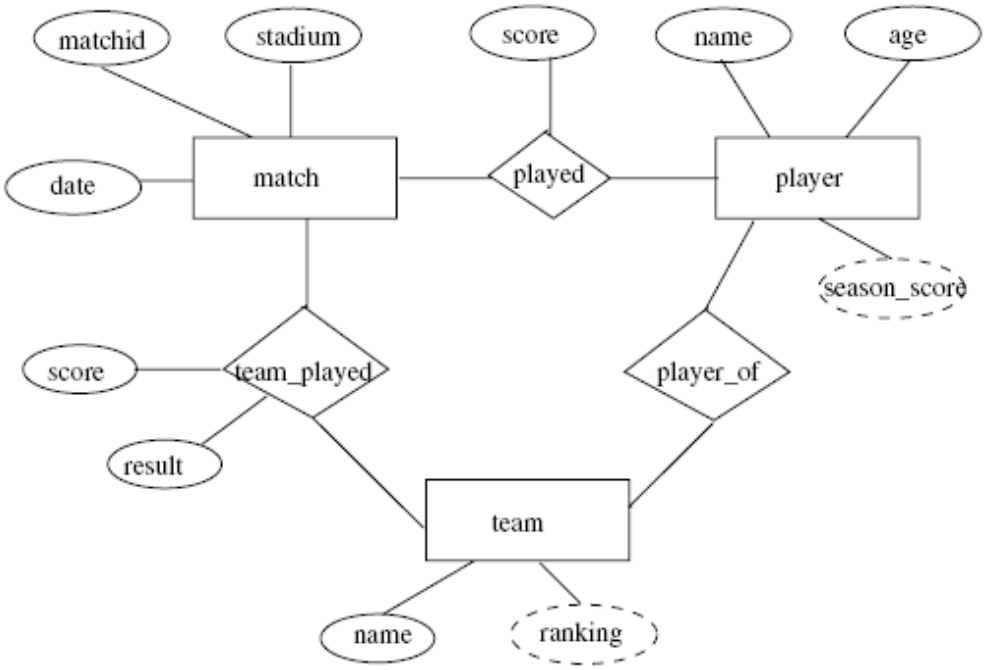


Figure 2.7 E-R diagram for all teams statistics.

Consider a university database for the scheduling of classrooms for final exams. This database could be modeled as the single entity set *exam*, with attributes *course-name*, *section-number*, *room-number*, and *time*. Alternatively, one or more additional entity sets could be defined, along with relationship sets to replace some of the attributes of the *exam* entity set, as

- *course* with attributes *name*, *department*, and *c-number*
- *section* with attributes *s-number* and *enrollment*, and dependent as a weak entity set on *course*
- *room* with attributes *r-number*, *capacity*, and *building*

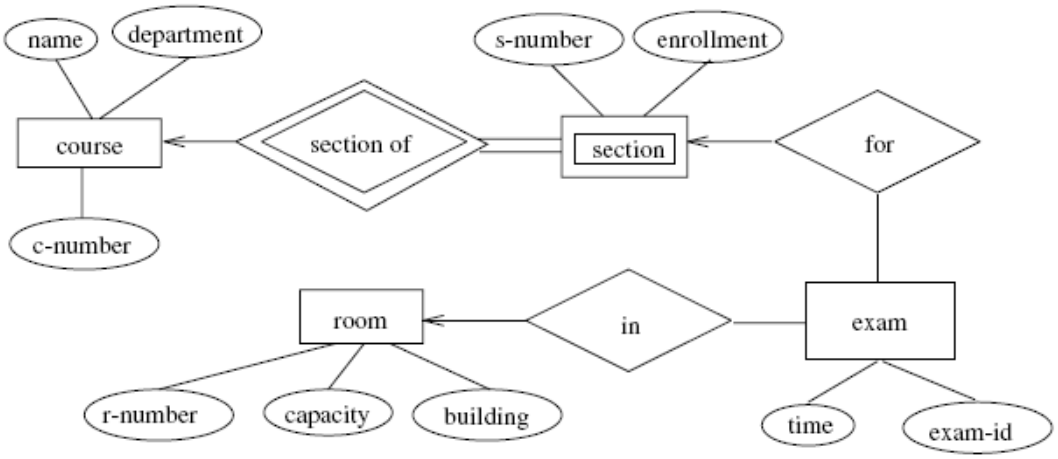
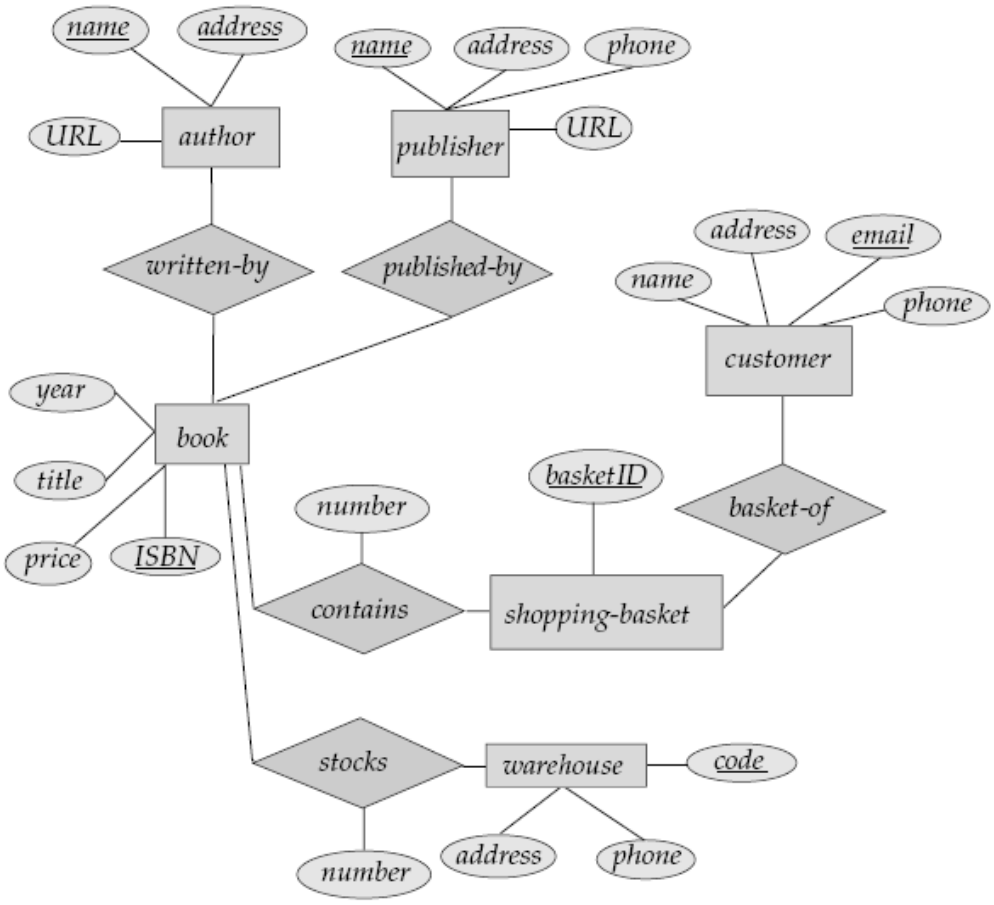


Figure 2.12 E-R diagram for exam scheduling.



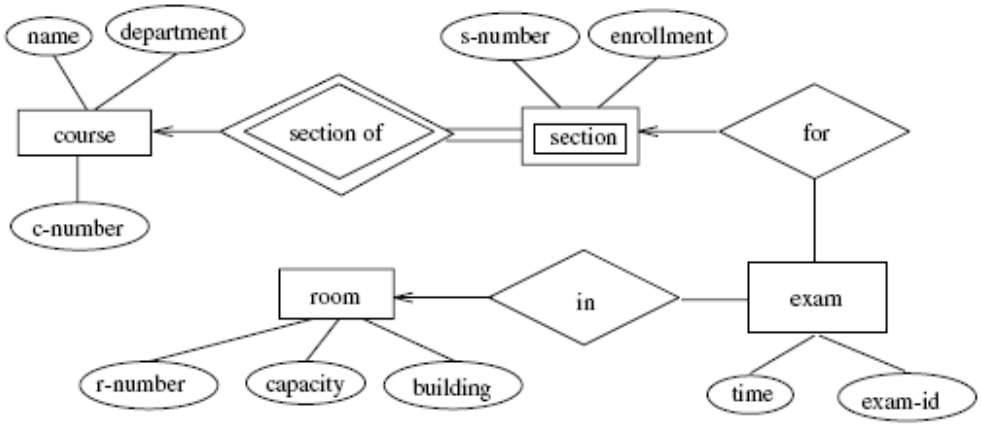


Figure 2.12 E-R diagram for exam scheduling.

Advanced ER-diagram:

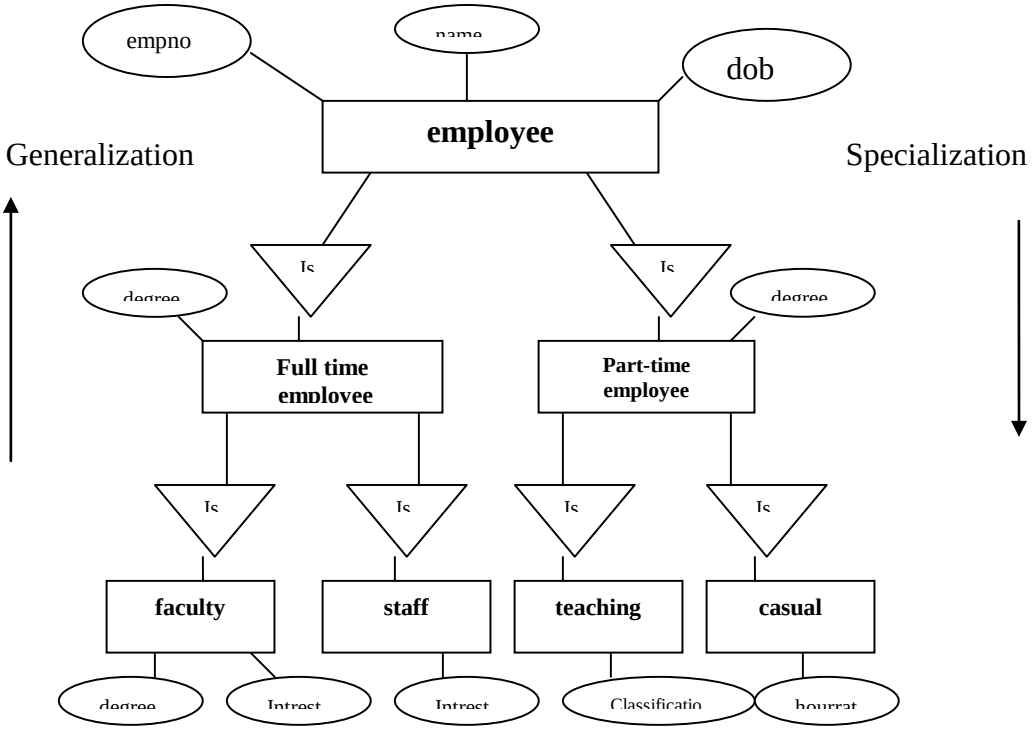
Abstraction is the simplification mechanism used to hide superfluous details of a set of objects. It allows one to concentrate on the properties that are of interest to the application.

There are two main abstraction mechanism used to model information:

Generalization and specialization:

. *Generalization* is the abstracting process of viewing set of objects as a single general class by concentrating on the general characteristics of the constituent sets while suppressing or ignoring their differences. It is the union of a number of lower-level entity types for the purpose of producing a higher-level entity type. For instance, student is a generalization of graduate or undergraduate, full-time or part-time students. Similarly, employee is generalization of the classes of objects cook, waiter, and cashier. Generalization is an IS_A relationship; therefore, manager IS_AN employee, cook IS_AN employee, waiter IS_AN employee, and so forth.

Specialization is the abstracting process of introducing new characteristics to an existing class of objects to create one or more new classes of objects. This involves taking a higher-level, and using additional characteristics, generating lower-level entities. The lower-level entities also inherits the, characteristics of the higher-level entity. In applying the characteristics size to car we can create a full-size, mid-size, compact or subcompact car. Specialization may be seen as the reverse process of generalization addition specific properties are introduced at a lower level in a hierarchy of objects.



EMPLOYEE(**empno**,name,dob)
FULL_TIME_EMPLOYEE(**empno**,salary)
PART_TIME_EMPLOYEE(**empno**,type)

Faculty(**empno**,degree,intrest)
Staff(**empno**,hour-rate)
Teaching (**empno**,stipend)

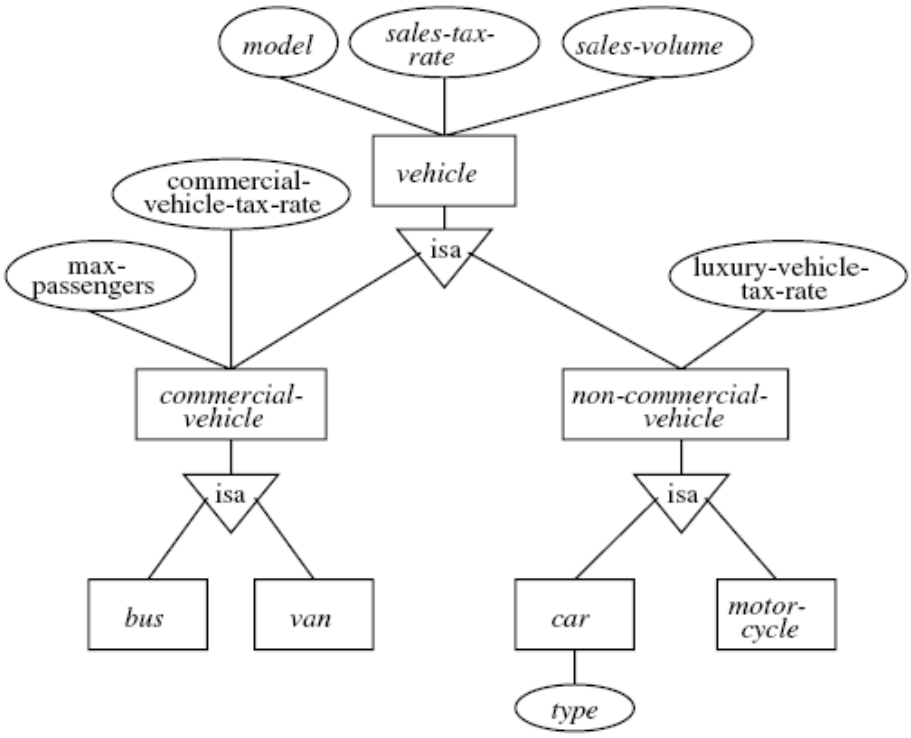
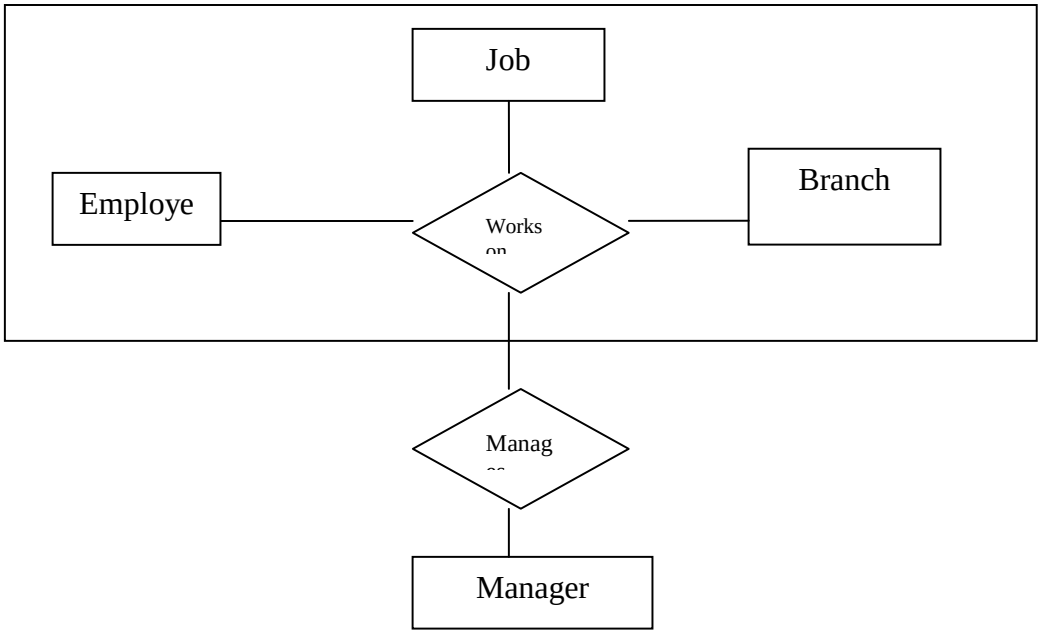


Figure 2.19 E-R diagram of motor-vehicle sales company.

Aggregation:

Aggregation is the process of compiling information on an object, there by abstracting a higher level object. In this manner, the entity person is derived by aggregating the characteristics of name, address, ssn. Another form of the aggregation is abstracting a relationship objects and viewing the relationship as an object.


Job
Employe
Works
Branch
Manag
Manager

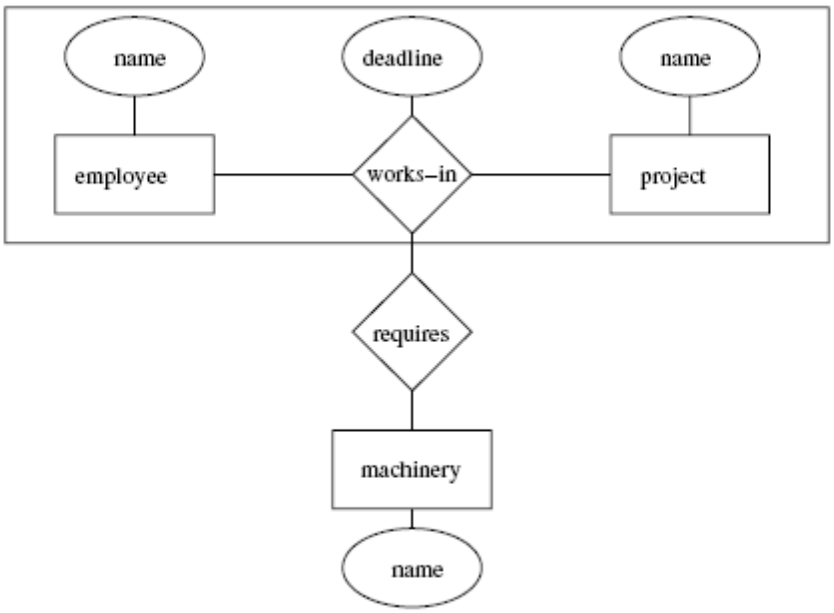


Figure 2.8 E-R diagram Example 1 of aggregation.

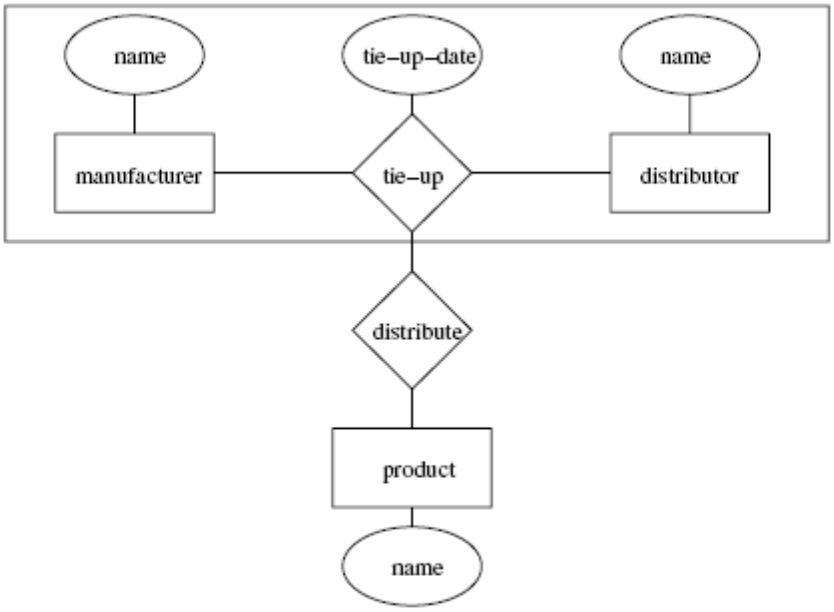
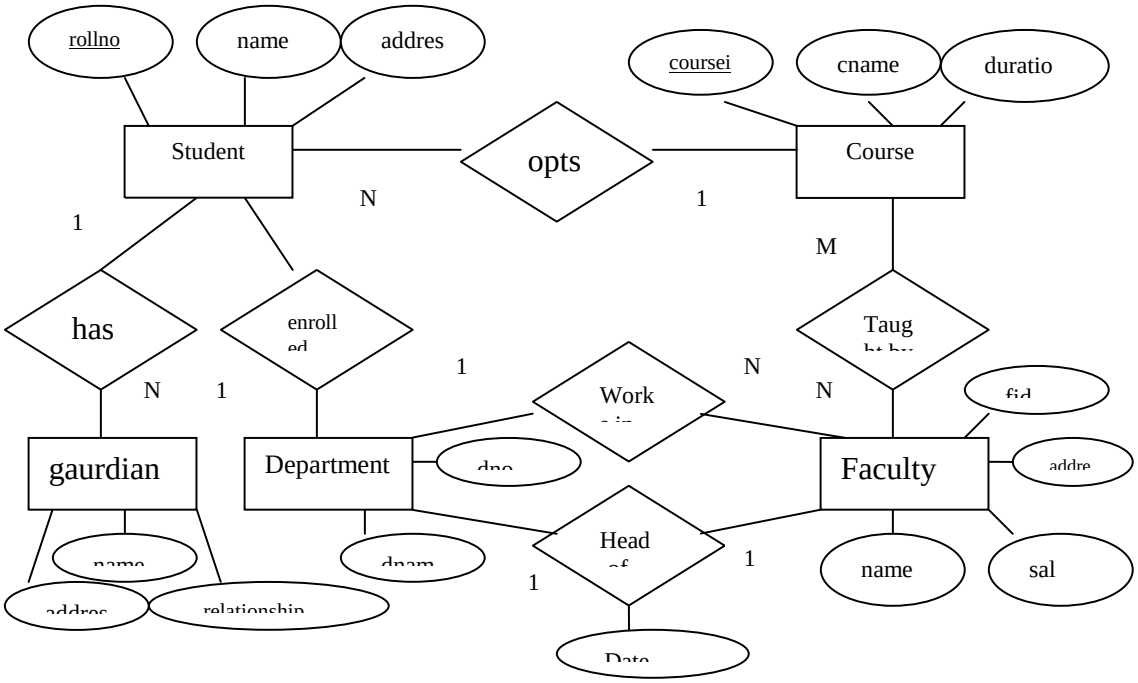


Figure 2.9 E-R diagram Example 2 of aggregation.

Explain the distinctions among the terms primary key, candidate key, and superkey.

Answer: A *superkey* is a set of one or more attributes that, taken collectively, allows us to identify uniquely an entity in the entity set. A superkey may contain extraneous attributes. If *K* is a superkey, then so is any superset of *K*. A superkey for which no proper subset is also a superkey is called a *candidate key*. It is possible that several distinct sets of attributes could serve as candidate keys. The *primary key* is one of the candidate keys that is chosen by the database designer as the principal means of identifying entities within an entity set.

ER- Diagram For College Database



Conversion of ER-diagram to relational database

Conversion of entity sets:

1. For each strong entity type E in the ER diagram, we create a relation R containing all the single attributes of E. The primary key of the relation R will be one of the key attribute of R.

   STUDENT(rollno (primary key),name, address)
   FACULTY(id(primary key),name ,address, salary)
   COURSE(course-id,(primary key),course_name,duration)
   DEPARTMENT(dno(primary key),dname)

2. for each weak entity type W in the ER diagram, we create another relation R that contains all simple attributes of W. If E is an owner entity of W then key attribute of E is also include In R. This key attribute of R is set as a foreign key attribute of R. Now the combination of primary key attribute of owner entity type and partial key of the weak entity type will form the key of the weak entity type

   GUARDIAN((rollno,name) (primary key),address,relationship)

Conversion of relationship sets:

Binary Relationships:

- **One-to-one relationship:**

For each 1:1 relationship type R in the ER-diagram involving two entities E1 and E2 we choose one of entities(say E1) preferably with total participation and add primary key attribute of another E as a foreign key attribute in the table of entity(E1). We will also include all the simple attributes of relationship type R in E1 if any, For example, the department relationship has been extended tp include head-id and attribute of the relationship.

DEPARTMENT(D_NO,D_NAME,HEAD_ID,DATE_FROM)

- **One-to-many relationship:**

For each 1:n relationship type R involving two entities E1 and E2, we identify the entity type (say E1) at the n-side of the relationship type R and include primary key of the entity on the other side of the relation (say E2) as a foreign key attribute in the table of E1. We include all simple attribute(or simple components of a composite attribute of R(if any) in he table E1)
For example:
The works in relationship between the DEPARTMENT and FACULTY. For this relationship choose the entity at N side, i.e, FACULTY and add primary key attribute of another entity DEPARTMENT, ie, DNO as a foreign key attribute in FACULTY.

FACULTY(CONSTAINS WORKS_IN RELATIOSHIP)
(ID,NAME,ADDRESS,BASIC_SAL,DNO)

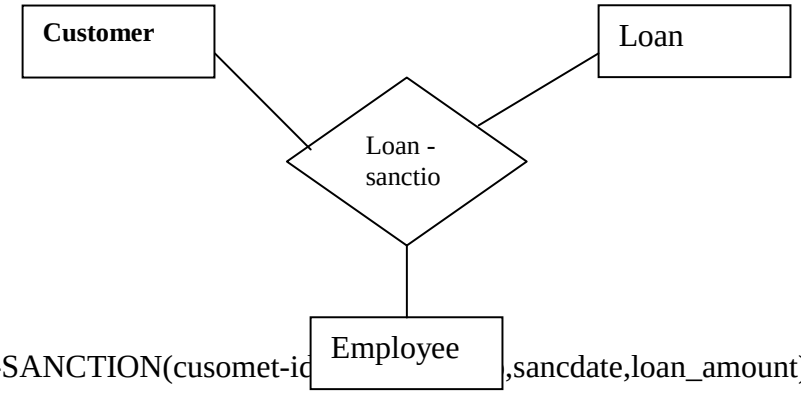
- **Many-to-many relationship:**

For each m:n relationship type R, we create a new table (say S) to represent R, We also include the primary key attributes of both the participating entity types as a foreign key attribute in s. Any simple attributes of the m:n relationship type(or simple components as a composite attribute) is also included as attributes of S.
For example:
The M:n relationship taught-by between entities COURSE; and FACULTY shod be represented as a new table. The structure of the table will include primary key of COURSE and primary key of FACULTY entities.

TAUGHT-BY(ID (primary key of FACULTY table),course-id (primary key of COURSE table)

- **N-ary relationship:**

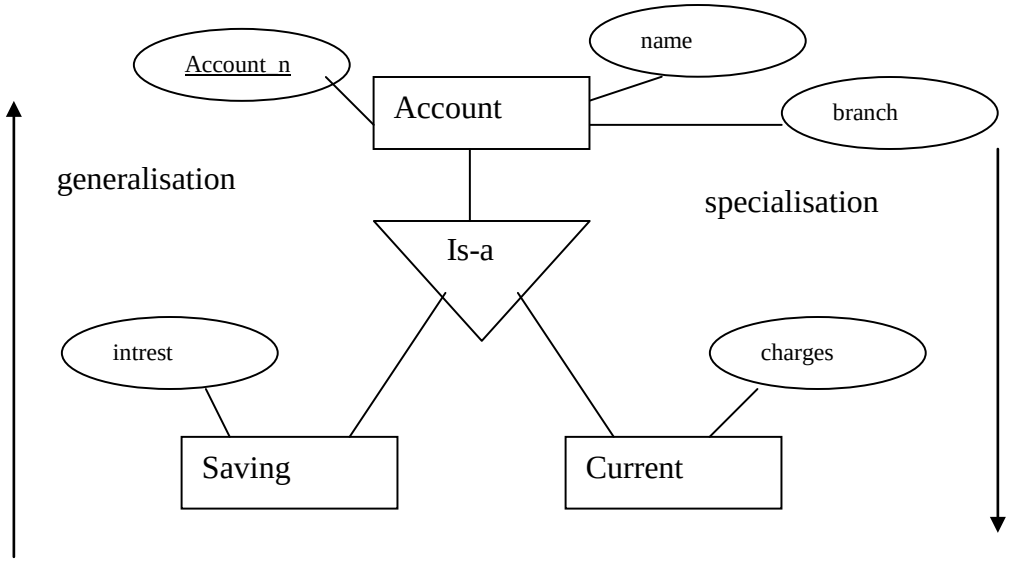
For each n-anry relationship type R where n>2, we create a new table S to represent R, We include as foreign key attributes in s the primary keys of the relations that represent the participating entity types. We also include any simple attributes of the n-ary relationship type(or simple components of complete attribute) as attributes of S. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types.



LOAN-SANCTION(cusomet-id ,sancdate,loan_amount)

- **Multi-valued attributes:**

For each multivalued attribute ‘A’, we create a new relation R that includes an attribute corresponding to plus the primary key attributes k of the relation that represents the entity type or relationship that has as an attribute. The primary key of R is then combination of A and k.
For example, if a STUDENT entity has rollno,name and phone number where phone numer is a multivalued attribute the we will create table PHONE(rollno,phoneno) where primary key is the combination,In the STUDENT table we need not have phone number, instead if can be simply (rollno,name) only.
PHONE(rollno,phoneno)



- **Converting Generalisation /specification hierarchy to tables:**

A simple rule for conversion may be to decompose all the specialized entities into table in case they are disjoint, for example, for the figure we can create the two table as:

Account(account_no,name,branch,balance)
Saving account(account-no,intrest)
Current_account(account-no,charges)

Record Based Logical Model

Hierarchical Model:

- A hierarchical database consists of a collection of *records* which are connected to one another through *links.*
- a record is a collection of fields, each of which contains only one data value.
- A link is an association between precisely two records.
- The hierarchical model differs from the network model in that the records are organized as collections of trees rather than as arbitrary graphs.

Tree-Structure Diagrams:

- The schema for a hierarchical database consists of
  - *boxes,* which correspond to record types
  - *lines,* which correspond to links
- Record types are organized in the form of a *rooted tree.*
  - No cycles in the underlying graph.
  - Relationships formed in the graph must be such that only one-to-many or one-to-one relationships exist between a parent and a child.

Database schema is represented as a collection of tree-structure diagrams.

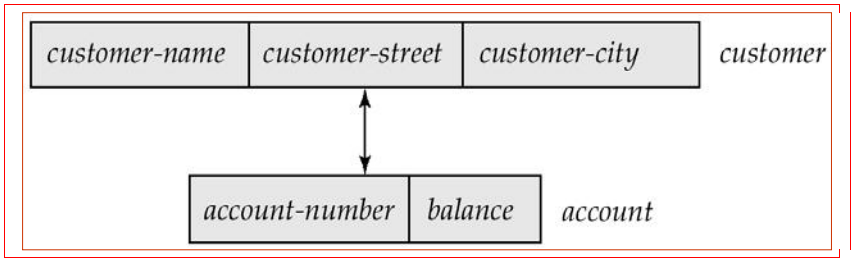
- *single* instance of a database tree
- The root of this tree is a dummy node
- The children of that node are actual instances of the appropriate record type

When transforming E-R diagrams to corresponding tree-structure diagrams, we must ensure that the resulting diagrams are in the form of rooted trees.

Single Relationships:

- Example E-R diagram with two entity sets, *customer* and *account,* related through a binary, one-to-many relationship *depositor.*
- Corresponding tree-structure diagram has
  - the record type *customer* with three fields: *customer-name, customer-street,* and *customer-city.*
  - the record type *account* with two fields: *account-number* and *balance*
  - the link *depositor,* with an arrow pointing to *customer*

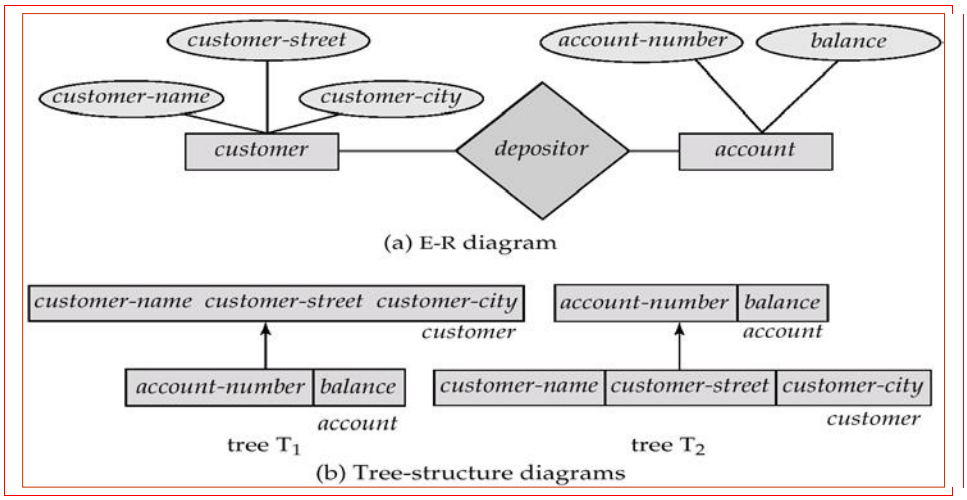
- If the relationship *depositor* is one to one, then the link *depositor* has two arrows.



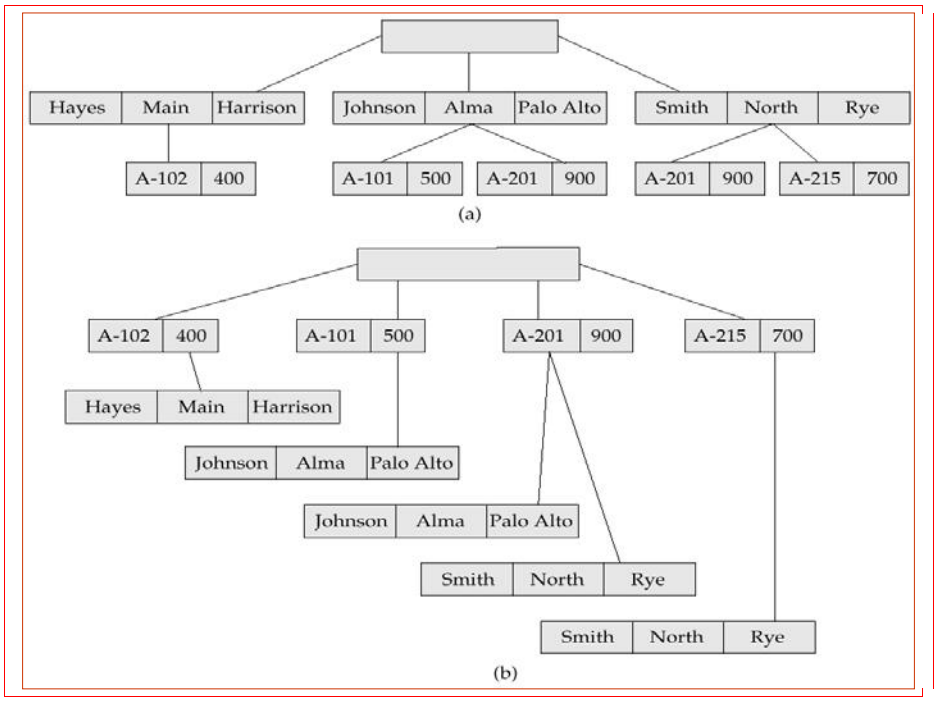
- Only one-to-many and one-to-one relationships can be directly represented in the hierarchical mode.

Transforming Many-To-Many Relationships:

- Must consider the type of queries expected and the degree to which the database schema fits the given E-R diagram.
- In all versions of this transformation, the underlying database tree (or trees) will have replicated records.

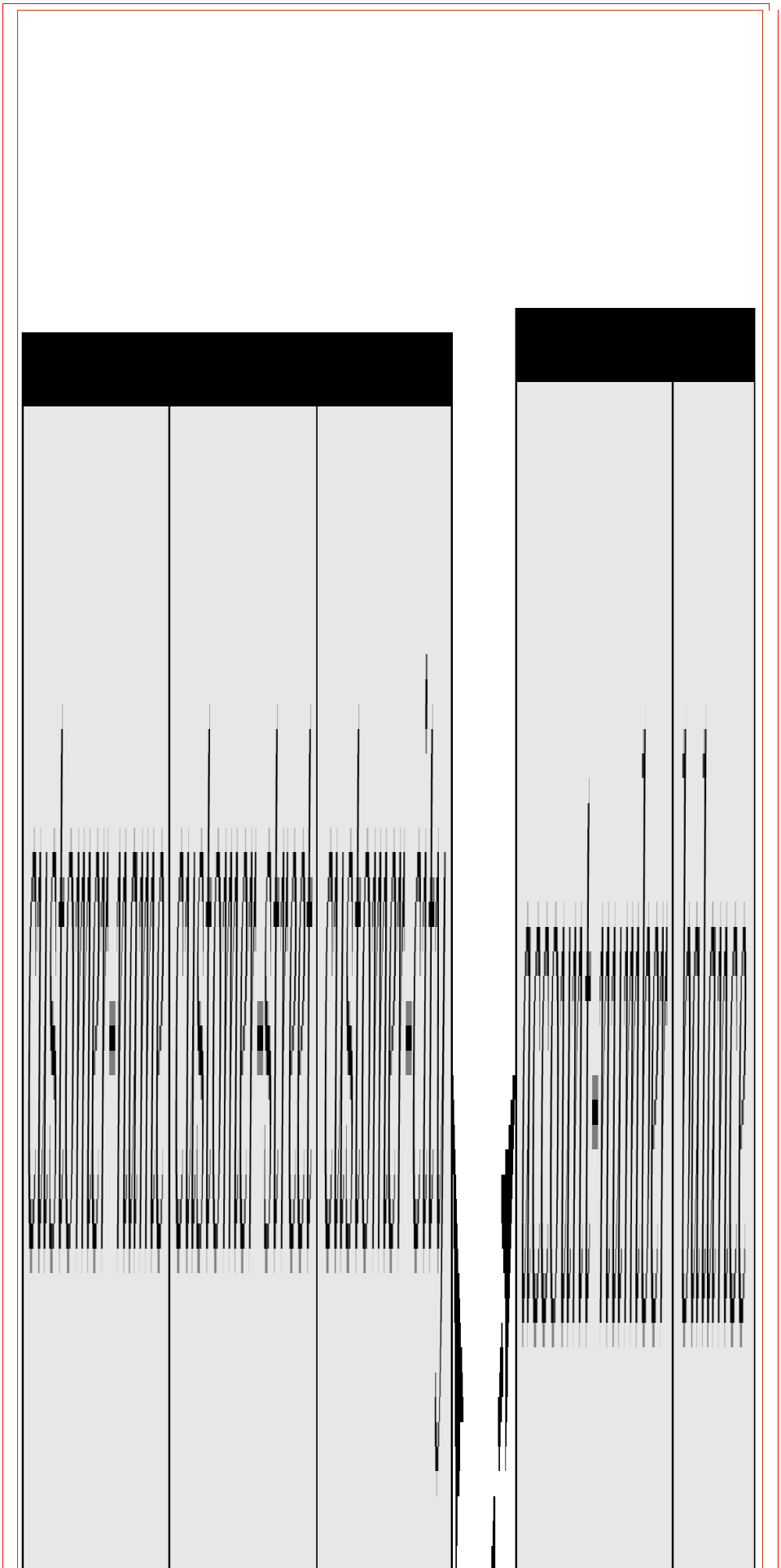


- Create two tree-structure diagrams, *T*1, with the root *customer*, and *T*2, with the root *account.*
- In *T*1, create *depositor*, a many-to-one link from *account* to *customer.*
- In *T*2, create *account-customer*, a many-to-one link from *customer* to *account.*



Virtual Records:

- For many-to-many relationships, record replication is necessary to preserve the tree-structure organization of the database.
  - Data inconsistency may result when updating takes place
  - Waste of space is unavoidable
- *Virtual record* — contains no data value, only a logical pointer to a particular physical record.
- When a record is to be replicated in several database trees, a single copy of that record is kept in one of the trees and all other records are replaced with a virtual record.
- Let *R* be a record type that is replicated in *T*1, *T*2, . . ., *Tn*. Create a new virtual record type *virtual-R* and replace *R* in each of the *n* – 1 trees with a record of type *virtual-R.*
- Eliminate data replication in the diagram shown on page B.11; create *virtual-customer* and *virtual-account.*
- Replace *account* with *virtual-account* in the first tree, and replace *customer* with *virtual-customer* in the second tree.
- Add a dashed line from *virtual-customer* to *customer*, and from *virtual-account* to *account*, to specify the association between a virtual record and its corresponding physical record.



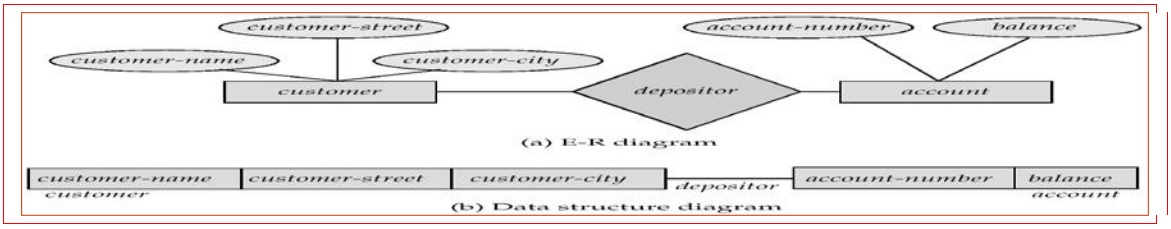
Network Model:

- Data are represented by collections of *records*.
  - similar to an entity in the E-R model
  - Records and their fields are represented as *record type*
- type *customer* = record
  customer-name: string;
  customer-street: string;
  customer-city: string;
  type *account* = record type
  account-number: integer;
  balance: integer;
- end
  end
- Relationships among data are represented by *links*
  - similar to a restricted (binary) form of an E-R relationship
  - restrictions on links depend on whether the relationship is many-many, many-to-one, or one-to-one.

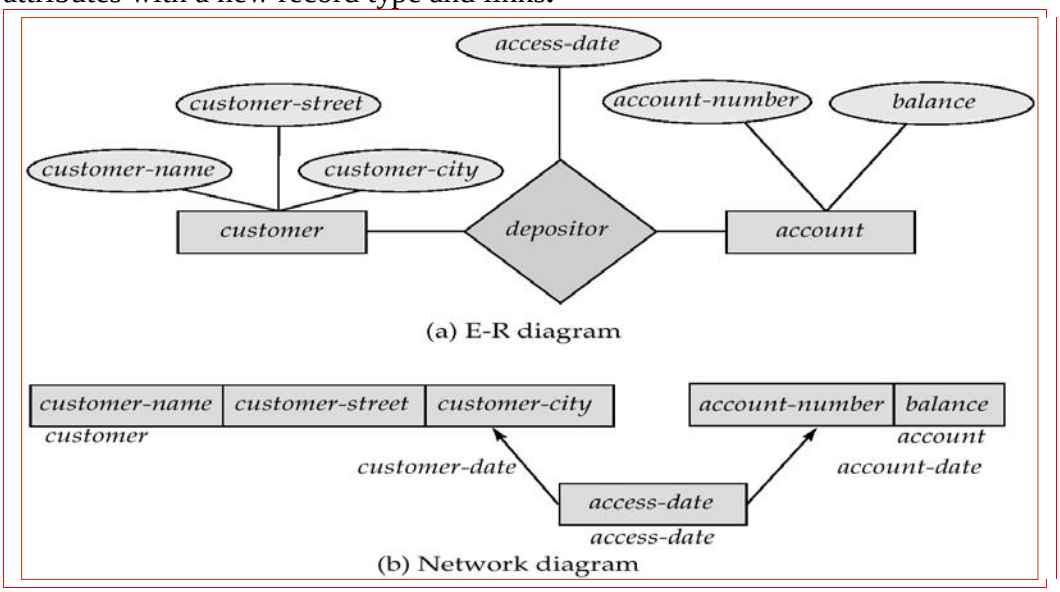
Data-Structure Diagrams:

- Schema representing the design of a network database.
- A data-structure diagram consists of two basic components:
  - Boxes, which correspond to record types.
  - Lines, which correspond to links.
- Specifies the overall logical structure of the database.

For every E-R diagram, there is a corresponding data-structure diagram.

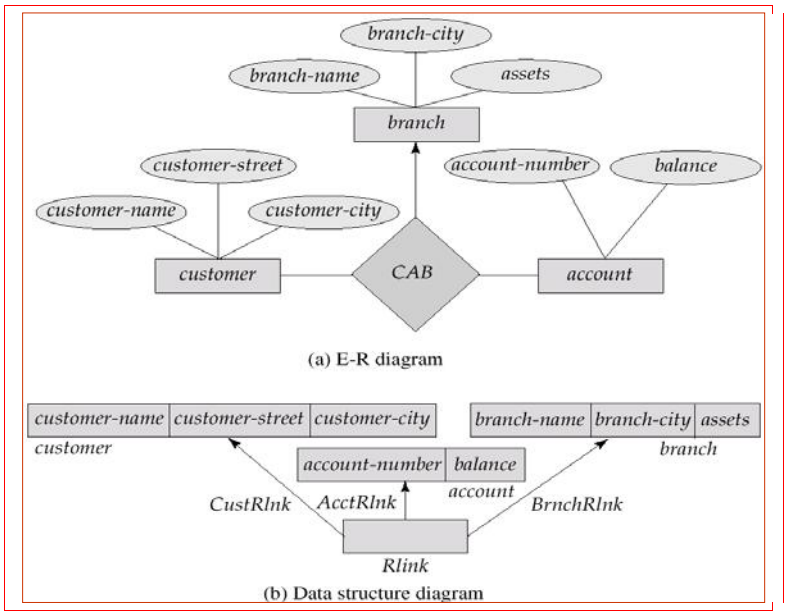


Since a link cannot contain any data value, represent an E-R relationship with attributes with a new record type and links.



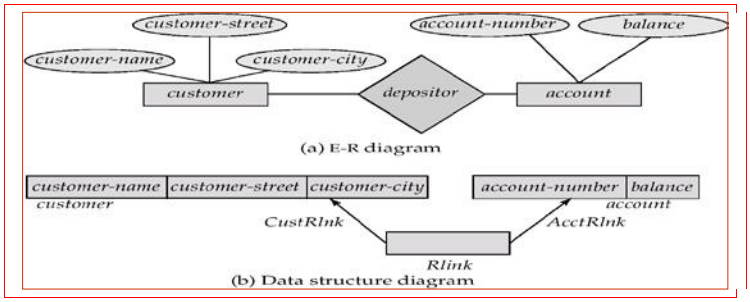
To represent an E-R relationship of degree 3 or higher, connect the participating record types through a new record type that is linked directly to each of the original record types.

1. Replace entity sets *account*, *customer*, and *branch* with record types *account*, *customer*, and *branch*, respectively.
2. Create a new record type *Rlink* (referred to as a *dummy* record type).
3. Create the following many-to-one links:
   - *CustRlink* from *Rlink* record type to *customer* record type
   - *AcctRlnk* from *Rlink* record type to *account* record type
   - *BrncRlnk* from *Rlink* record type to *branch* record type



The DBTG CODASYL Model:

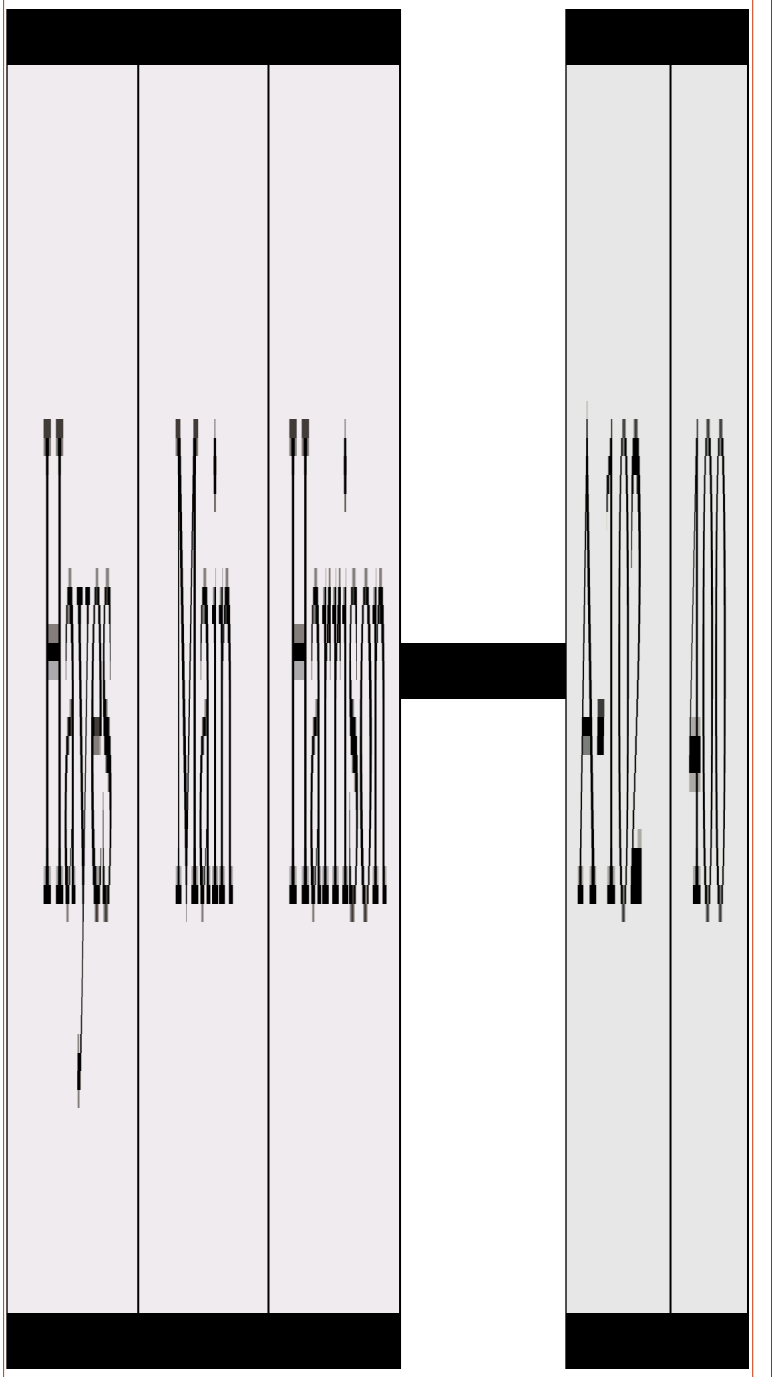
- All links are treated as many-to-one relationships.
- To model many-to-many relationships, a record type is defined to represent the relationship and two links are used.

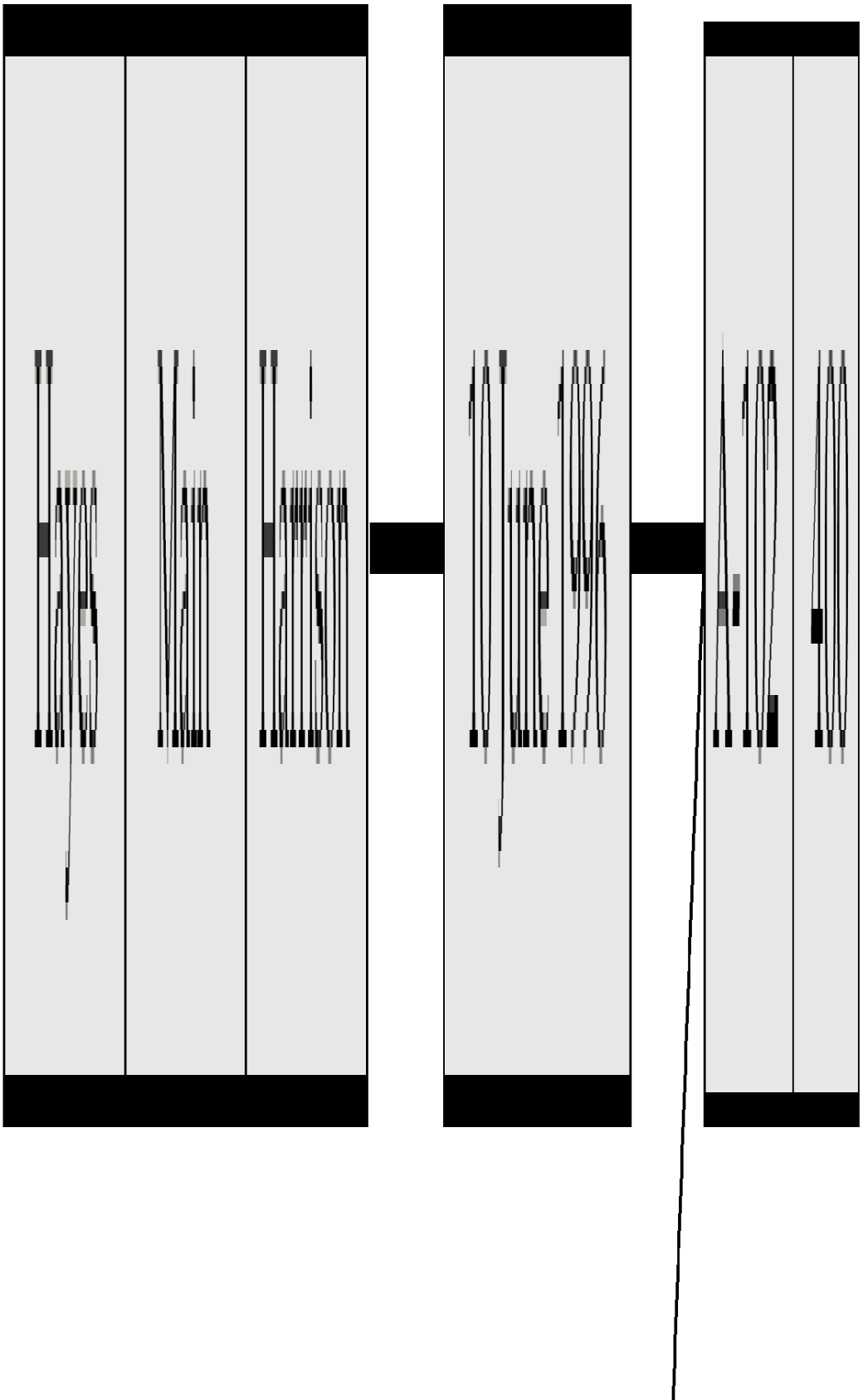


DBTG Sets:

- The structure consisting of two record types that are linked together is referred to in the DBTG model as a *DBTG set*
- In each DBTG set, one record type is designated as the *owner*, and the other is designated as the *member*, of the set.
- Each DBTG set can have any number of *set occurrences* (actual instances of linked records).
- Since many-to-many links are disallowed, each set occurrence has precisely one owner, and has zero or more member records.
- No member record of a set can participate in more than one occurrence of the set at any point.
- A member record can participate simultaneously in several set occurrences of *different* DBTG sets.

o





RELATIONAL MODEL

Relational model is simple model is simple model in which database is represented as a collection of “relations” where each relation is represented by two-dimensional table.

account_number	*branch_name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

The relational model was founded by E.F.Codd of the IBM in 1972.The basic concept in the relational model is that of a relation.

Properties:

- It is column homogeneous. In other words, in any given column of a table, all items are of the same kind.
- Each item is a simple number or a character string. That is a table must be in first normal form.
- All rows of a table are distinct.
- The ordering of rows with in a table is immaterial.

- The column of a table are assigned distinct names and the ordering of these columns in immaterial.

Domain, attributes tuples and relational:

Tuple:

Each row in a table represents a record and is called a tuple .A table containing 'n' attributes in a record is called is called n-tuple.

Attributes:

The name of each column in a table is used to interpret its meaning and is called an attribute.Each table is called a relation.

In the above table, account_number, branch name, balance are the attributes.

Domain:

A domain is a set of values that can be given to an attributes. So every attribute in a table has a specific domain. Values to these attributes can not be assigned outside their domains.

Relation:

A relation consist of

- **Relational schema**
- **Relation instance**

Relational schema:

A relational schema specifies the relation' name, its attributes and the domain of each attribute. If R is the name of a relation and A1,A2,… and is a list of attributes representing R then R(A1,A2,…,an) is called a relational schema. Each attribute in this relational schema takes a value from some specific domain called domain(Ai).

Example:

PERSON(PERSON_IDinteger,NAME: STRING,AGE:INTEGER,ADDRESS:string)

Total number of attributes in a relation denotes the degree of a relation.since the PERSON relation schemea contains four attributes ,so this relation is of degree 4.

Relation Instance:

A relational instance denoted as r is a collection of tuples for a given relational schema at a specific point of time.

A relation state r to the relations schema R(A1,A2…,An) also denoted by r® is a set of n-tuples

R{t1,t2,…tm}

Where each n-tuple is an ordered list of n values

T=<v1,v2,….vn>

Where each vi belongs to domain (Ai) or contains null values.

The relation schema is also called 'intension' and the relation state is also called 'extension'.

Eg:

Relation schema for student:

STUDENT(rollno:strinhg,name:string,city:string,age:integer)

Relation instance:
Student:

Rollno	Name	City	Age
101	Sujit	Bam	23
102	kunal	bbsr	22

Keys:

Super key:

A super key is an attribute or a set of attributes used to identify the records uniquely in a relation.

For example , customer-id,(cname,customer-id),(cname,telno)

Candidate key:

Super keys of a relation can contain extra attributes . candidate keys are minimal super keys. i.e, such a key contains no extraneous attribute. An attribute is called extraneous if even after removing it from the key, makes the remaining attributes still has the properties of a key.

In a relation R, a candidate key for R is a subset of the set of attributes of R, which have the following properties:

- *Uniqueness:* no two distinct tuples in R have the same values for the candidate key
- *Irreducible:* No proper subset of the candidate key has the *uniqueness property that is the candidate key.*
- *A candidate key's values must exist. It can't be null.*
- *The values of a candidate key must be stable. Its value can not change outside the control of the system.*

Eg: (cname,telno)

Primary key:

The primary key is the candidate key that is chosen by the database designer as the principal means of identifying entities with in an entity set. The remaining candidate keys if any are called ***alternate key*****.**

RELATIONAL CONSTRAINTS:

There are three types of constraints on relational database that include

- DOMAIN CONSTRAINTS
- KEY CONSTRAINTS
- INTEGRITY CONSTRAINTS

DOMAIN CONSTRAINTS:

It specifies that each attribute in a relation an atomic value from the corresponding domains. The data types associated with commercial RDBMS domains include:

- Standard numeric data types for integer
- Real numbers
- Characters
- Fixed length strings and variable length strings

Thus, domain constraints specifies the condition that we to put on each instance of the relation. So the values that appear in each column must be drawn from the domain associated with that column.

Rollno	Name	City	Age
101	Sujit	Bam	23
102	kunal	bbsr	22

Key constraints:
This constraints states that the key attribute value in each tuple msut be unique .i.e, no two tuples contain the same value for the key attribute.(null values can allowed)

Emp(empcode,name,address) . here empcode can be unique

Integrity constraints:

There are two types of integrity constraints:

- Entity integrity constraints
- Referential integrity constraints

Entity integrity constraints:
It states that no primary key value can be null and unique. This is because the primary key is used to identify individual tuple in the relation. So we will not be able to identify the records uniquely containing null values for the primary key attributes. This constraint is specified on one individual relation.

Referential integrity constraints:
It states that the tuple in one relation that refers to another relation must refer to an existing tuple in that relation. This constraints is specified on two relations .
If a column is declared as foreign key that must be primary key of another table.

Department(deptcode,dname)
Here the deptcode is the primary key.
Emp(*empcode*,name,city,*deptcode*).
Here the deptcode is foreign key.

CODD'S RULES

Rule 1 : The information Rule.

"All information in a relational data base is represented explicitly at the logical level and in exactly one way - by values in tables."

Everything within the database exists in tables and is accessed via table access routines.

Rule 2 : Guaranteed access Rule.

"Each and every datum (atomic value) in a relational data base is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name."

To access any data-item you specify which column within which table it exists, there is no reading of characters 10 to 20 of a 255 byte string.

Rule 3 : Systematic treatment of null values.

"Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type."

If data does not exist or does not apply then a value of NULL is applied, this is understood by the RDBMS as meaning non-applicable data.

Rule 4 : Dynamic on-line catalog based on the relational model.

"The data base description is represented at the logical level in the same way as-ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data."

The Data Dictionary is held within the RDBMS, thus there is no-need for off-line volumes to tell you the structure of the database.

Rule 5 : Comprehensive data sub-language Rule.

"A relational system may support several languages and various modes of terminal use (for example, the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all the following items

- Data Definition
- View Definition
- Data Manipulation (Interactive and by program).
- Integrity Constraints
- Authorization.

Every RDBMS should provide a language to allow the user to query the contents of the RDBMS and also manipulate the contents of the RDBMS.

Rule 6 : .View updating Rule

"All views that are theoretically updateable are also updateable by the system."

Not only can the user modify data, but so can the RDBMS when the user is not logged-in.

Rule 7 : High-level insert, update and delete.

"The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data."

The user should be able to modify several tables by modifying the view to which they act as base tables.

Rule 8 : Physical data independence.

"Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods."

The user should not be aware of where or upon which media data-files are stored

Rule 9 : Logical data independence.

"Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit un-impairment are made to the base tables."

User programs and the user should not be aware of any changes to the structure of the tables (such as the addition of extra columns).

Rule 10 : Integrity independence.

"Integrity constraints specific to a particular relational data base must be definable in the relational data sub-language and storable in the catalog, not in the application programs."

If a column only accepts certain values, then it is the RDBMS which enforces these constraints and not the user program, this means that an invalid value can never be entered into this column, whilst if the constraints were enforced via programs there is always a chance that a buggy program might allow incorrect values into the system.

Rule 11 : Distribution independence.

"A relational DBMS has distribution independence."

The RDBMS may spread across more than one system and across several networks, however to the end-user the tables should appear no different to those that are local.

Rule 12 : Non-subversion Rule.

"If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity Rules and constraints expressed in the higher level relational language (multiple-records-at-a-time)."

RELATION ALGEBRA:

Relational algebra is a set of basic operations used to manipulate the data in relational model. These operations enable the user to specify basic retrieval request. The result of retrieval is anew relation, formed from one or more relation. These operation can be classified in two categories.

- **Basic Set Operation**
  - Union
  - Intersection
  - Set difference
  - Cartesian product
- **Relational operations**
  - Select
  - Project
  - Join
  - Division

Basic set operation:

These are the binary operations; i.e, each is applied to two sets or relations. These two relations should be union compatible except in case of Cartesian product.

Two relations R(A1,A2…,AN) and S(B1,B2…BN) are said to be union compatible if they have the same degree n and domains of the corresponding attributes are also the same; domain(Ai)=Domain(bI) for 1<=i<=n.

Union Operation – Example

Relations *r, s:*

r ∪ s:

A	*B*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

A	B
α	1
α	2
β	1
β	3

UNION OPERATION:

Notation: $r \cup s$

Defined as:

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

For $r \cup s$ to be valid.

***r, s* must have the *same arity* (same number of attributes)**

2. The attribute domains must be *compatible* (e.g., 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)

E.g. to find all customers with either an account or a loan $\Pi_{customer\text{-}name}$ ***(depositor)*** $\cup \Pi_{customer\text{-}name}$ ***(borrower)***

Set Difference

Operation – Example

Relations *r, s:*

***r – s*:**

A	B

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

α	1
β	1

Set Difference Operation

- **Notation *r – s***
- **Defined as:**
  - ***r – s* = {*t* | *t* ∈ *r* and t ∉ *s*}**
- **Set differences must be taken between *compatible* relations.**
  - ***r* and *s* must have the *same arity***
  - **attribute domains of *r* and *s* must be compatible**

Cartesian product – Example

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

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

r x s:

Project Operation – Example

Relation *r*:

A	C
α	1
α	1
β	1
β	2

$\Pi_{A,C}(r)$

A	C
α	1
β	1
β	2

Notation *r* x *s*

Defined as:

***r* x *s* = {*t q* | *t* ∈ *r* and *q* ∈ *s*}**

Notation:

Assume that at[illegible] of r(R) [illegible](S) are disjoint. (That is, $R \cap S = \emptyset$).

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

where A_1, A_2 are attribute names and *r* is a relation name.

If attributes of *r*(R), *s*(S) a[illegible] disjoint, then renaming must be used.

- **The result is defined as the relation of *k* columns obtained by erasing the columns that are not listed**
- **Duplicate rows removed from result, since relations are sets**
- **E.g. To eliminate the *branch-name* attribute of *account***

  $\Pi_{account\text{-}number,\ balance}$ **(*account*)**

Select Operation – Example

- Relation *r*

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Notation: $\sigma_p(r)$

***p* is called the selection predicate**

Defined as:

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

Where *p* is a formula in propositional calculus consisting of terms connected by : $\wedge$ (and), $\vee$ (or), $\neg$ (not) Each term is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: =, ≠, >, ≥. <. ≤

Example of selection:

Q: Display the account details belonging to the branch "perryridge". $\sigma_{branch\text{-}name=\text{“}Perryridge\text{”}}(account)$

Rename Operation

Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
Allows us to refer to a relation by more than one name.
Example:

$$\rho_X(E)$$

returns the expression *E* under the name *X*
If a relational-algebra expression *E* has arity *n*, then

$$\rho_{X(A1, A2, \ldots, An)}(E)$$

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

Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \text{ and } t \in s \}$
- Assume:
  - *r*, *s* have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$

Set-Intersection Operation - Example

- Relation r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r ∩ s

A	B
α	2

Prepared by: Dr. Subhendu Kumar Rath

Natural-Join Operation

- Notation: r ⋈ s
- Let r and s be relations on schemas R and S respectively. Then, r ⋈ s is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from r and t_s from s.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple t to the result, where
    - t has the same value as t_r on r
    - t has the same value as t_s on s
- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$
  - Result schema = (A, B, C, D, E)
  - $r \bowtie s$ is defined as:

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

Natural Join Operation – Example

- **Relations r. s:**

r

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

s

B	D	E
1	a	α
3	a	β
1	b	γ
2	a	δ
3	b	∈

$r \bowtie s$

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

Division Operation

$$r \div s$$

Suited to queries that include the phrase "for all".
Let *r* and *s* be relations on schemas R and S respectively where

- $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
- $S = (B_1, \ldots, B_n)$
- **The result of r ÷ s is a relation on schema**
- $R - S = (A_1, \ldots, A_m)$

$$r \div s = \{ t \mid t \in \Pi_{R-S}(r) \wedge \forall u \in s (tu \in r) \}$$

Division Operation – Example

Relations *r*, *s*:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
∈	6
∈	1
β	2

r

B
1
2

s

r ÷ *s*:

A
α
β

Prepared by: Dr. Subhendu Kumar Rath

Example Queries

- Find all customers who have an account from at least the "Downtown" and the Uptown" branches.

Query 1

$$\Pi_{CN}(\sigma_{BN="Downtown"}(depositor \bowtie account)) \cap \Pi_{CN}(\sigma_{BN="Uptown"}(depositor \bowtie account))$$

where *CN* denotes customer-name and *BN* denotes *branch-name*.

Query 2

$$\Pi_{customer\text{-}name,\ branch\text{-}name}(depositor \bowtie account) \div \rho_{temp(branch\text{-}name)}(\{("Downtown"), ("Uptown")\})$$

Example Queries

- Find all customers who have an account at all branches located in Brooklyn city.

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

Banking Example

branch (branch-name, branch-city, assets)

customer (customer-name, customer-street, customer-only)

account (account-number, branch-name, balance)

loan (loan-number, branch-name, amount)

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

Example Queries

- Find all loans of over $1200

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

- Find the loan number for each loan of an amount greater than $1200

$$\Pi_{loan\text{-}number}(\sigma_{amount > 1200}(loan))$$

Example Queries

- Find the names of all customers who have a loan, an account, or both, from the bank

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

- Find the names of all customers who have a loan and an account at bank.

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

Example Queries

Find the names of all customers who have a loan at the Perryridge branch.

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=\text{“}Perryridge\text{”}}\ (\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower\ x\ loan)))$$

Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name\ =\ \text{“}Perryridge\text{”}}\ (\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(\text{borrower x loan}))) - \Pi_{customer\text{-}name}(\text{depositor})$$

Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.
  - Query 1

$$\Pi_{\text{customer-name}}(\sigma_{\text{branch-name = “Perryridge”}}\ (\sigma_{\text{borrower.loan-number = loan.loan-number}}(\text{borrower x loan})))$$

  - Query 2

$$\Pi_{\text{customer-name}}(\sigma_{\text{loan.loan-number = borrower.loan-number}}((\sigma_{\text{branch-name = “Perryridge”}}(\text{loan}))\ x\ \text{borrower}))$$

Example Queries

Find the largest account balance

- Rename *account* relation as *d*
- The query is:

$$\Pi_{balance}(account) - \Pi_{account.balance}\ (\sigma_{account.balance\ <\ d.balance}\ (account\ x\ \rho_{d}\ (account)))$$

Aggregate functions:

Aggregation function takes a collection of values and returns a single value as a result.

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

Aggregate operation in relational algebra

$$G1, G2, \ldots, Gn\ \mathcal{G}\ F1(A1), F2(A2), \ldots, Fn(An)\ (E)$$

E is any relational-algebra expression
*G*1, *G*2 …, *G*n is a list of attributes on which to group (can be empty)
Each *Fi* is an aggregate function.
Each *Ai* is an attribute name

Find sum of sal fo all emp records

$\mathcal{G}_{sum(sal)}(emp)$

Find maximum salalry from emp table

$\mathcal{G}_{max(salary)}(emp)$

Find branch name and maximum salalry from emp table

$Branch\_name\ \mathcal{G}_{,max(salary)}(emp)$

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

$\mathcal{G}_{\mathbf{sum(c)}}(r)$

sum-C
27

OUTER JOIN:

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

There are three forms of outer join

- left outer join
- right outer join
- full outer join

employee:

Empname	Street	City
Coyote	Toon	Hollywood
Rabbit	Tunnel	carrot
Smith	Revolver	Death valley
William	Seaview	Seattle

Ft_works:

Empname	Branch name	Salary
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
William	Redmond	1500

Employee $\bowtie$ **ft_works**

Empname	Street	City	Branch name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	carrot	Mesa	1300
William	Seaview	Seattle	Redmond	1500

Left outer join:

It takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes. The right relation and adds them to the result of the natural join. In tuple (smith, Revolcer, Death valley, null, null) is such a tuple. All information from the left relation is present in the result of the left outer join.

Empname	Street	City	Branch name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	carrot	Mesa	1300
William	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death valley	Null	null

Result of Employee ⟕ ft_works

Right outer join:

It is symmetric with the left outer join. It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join. tuple(Gates,null,null,Redmond,5300) is such a tuple. Thus, all information from the right relation is present in the result of the right outer join.

Empname	Street	City	Branch name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	carrot	Mesa	1300
William	Seaview	Seattle	Redmond	1500
gates	Null	null	Redmond	5300

Full outer join:

It does both of those operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join. Figure 3.35 shows the result of a full outer join.

Since outer join operations may generate results containing null values, we need to specify how the different relation-algebra operations deal with null values. It is interesting to note that the outer join operations can be expressed by the basic relational algebra operations. For instance the left outer join operation

Employee ⟗ ft_works

Empname	Street	City	Branch name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	carrot	Mesa	1300
William	Seaview	Seattle	Redmond	1500
gates	Null	null	Redmond	5300

Tuple Relational Calculus

The tuple relational calculus is a non procedural query language. It describes the desired information with out giving a specific procedure for obtaining that information.
A query in the tuple relational calculus is expressed as:

{t|P(t)}

where P is a formula. Several tuple variables may appear in a formula. A tuple variable is said to be a free variable unless it is quantified by a ∃ or ∀.

- A nonprocedural query language, where each query is of the form

Example Queries

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

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

- Find the loan number for each loan of an amount greater than $1200

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

Notice that a relation on schema [*loan-number*] is implicitly defined by the query

Example Queries

Example Queries

Find the names of all customers having a loan, an account, or both at the bank

Find the names of all customers having a loan at the Perryridge branch

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

Find the names of all customers who have a loan and an account at the bank

$$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[customer\text{-}name]) \wedge \exists u \in depositor(t[customer\text{-}name] = u[customer\text{-}name])\}$$

Safety of Expressions

Find the names of all customers who have a loan and an account at the bank

$$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[customer\text{-}name]) \wedge \exists u \in depositor(t[customer\text{-}name] = u[customer\text{-}name])\}$$

It is possible to write tuple calculus expressions that generate infinite relations.

$$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[customer\text{-}$$

Domain Relational Calculus

A nonprocedural query language equivalent in power to the tuple relational calculus

Each query is an expression of the form:

Find the *loan-number*, *branch-name*, and *amount* for loans of

Example Queries

Query-by-Example (QBE)

QBE — Basic Structure

- A graphical query language which is based (roughly) on the domain relational calculus
- Two dimensional syntax – system creates templates of relations that are requested by users
- Queries are expressed “by example”

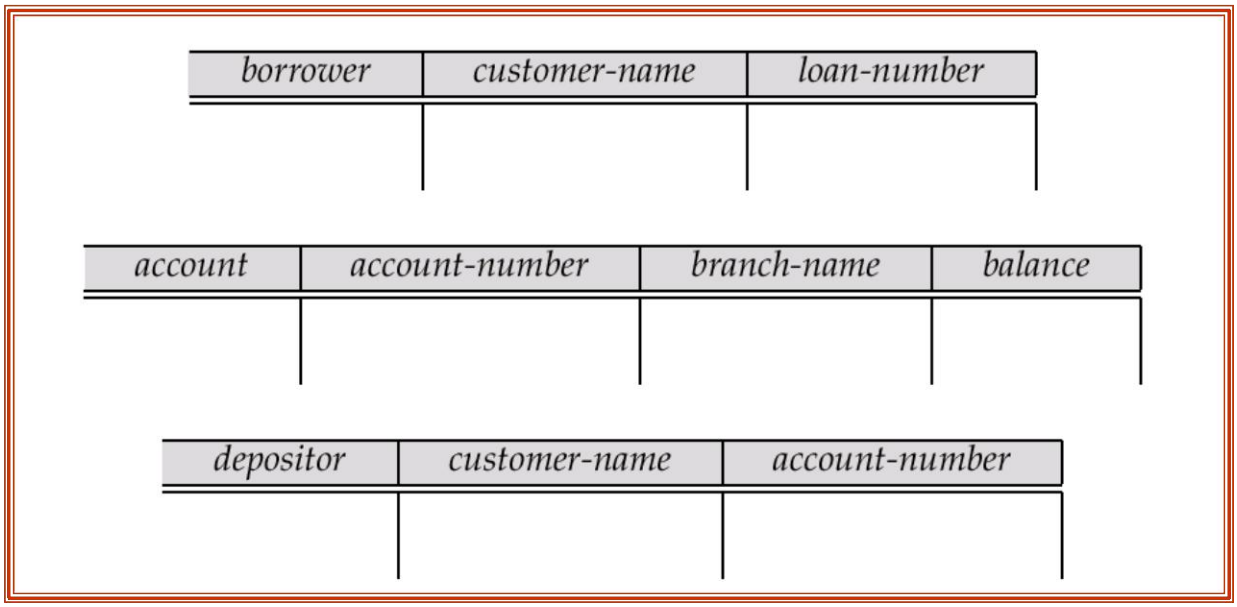
QBE Skeleton Tables for the Bank Example

branch	branch-name	branch-city	assets

customer	customer-name	customer-street	customer-city

loan	loan-number	branch-name	amount

QBE Skeleton Tables (Cont.)



Queries on One Relation

- Find all loan numbers at the Perryridge branch.

loan	loan-number	branch-name	amount
	P._x	Perryridge	

- _x is a variable (optional; can be omitted in above query)
- P. means print (display)
- duplicates are removed by default
- To retain duplicates use P.ALL.

loan	loan-number	branch-name	amount
	P.ALL.	Perryridge	

Queries on One Relation (Cont.)

- Display full details of all loans
  - Method 1:

loan	*loan-number*	*branch-name*	*amount*
	P._x	P._y	P._z

  - Method 2: Shorthand notation

loan	*loan-number*	*branch-name*	*amount*
P.			

Queries on One Relation (Cont.)

- Find the loan number of all loans with a loan amount of more than $700

loan	*loan-number*	*branch-name*	*amount*
	P.		>700

- Find names of all branches that are not located in Brooklyn

branch	*branch-name*	*branch-city*	*assets*
	P.	¬ Brooklyn	

Queries on One Relation (Cont.)

- Find the loan numbers of all loans made jointly to Smith and Jones.

borrower	*customer-name*	*loan-number*
	"Smith"	P._x
	"Jones"	_x

- Find all customers who live in the same city as Jones

customer	*customer-name*	*customer-street*	*customer-city*
	P._x		_y
	Jones		_y

Queries on Several Relations

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

loan	*loan-number*	*branch-name*	*amount*
	_x	Perryridge	

borrower	*customer-name*	*loan-number*
	P._y	_x

Queries on Several Relations (Cont.)

- Find the names of all customers who have both an account and a loan at the bank.

depositor	*customer-name*	*account-number*
	P._x	

borrower	*customer-name*	*loan-number*
	_x	

Negation in QBE

- Find the names of all customers who have an account at the bank, but do not have a loan from the bank.

depositor	*customer-name*	*account-number*
	P._x	

borrower	*customer-name*	*loan-number*
¬	_x	

Negation in QBE (Cont.)

- Find all customers who have at least two accounts.

depositor	*customer-name*	*account-number*
	P._x	_y
	_x	¬ _y

¬ means "not equal to"

The Condition Box

- Allows the expression of constraints on domain variables that are either inconvenient or impossible to express within the skeleton tables.
- Complex conditions can be used in condition boxes
- E.g. Find the loan numbers of all loans made to Smith, to Jones, or to both jointly

borrower	*customer-name*	*loan-number*
	_n	P._x

conditions
_n = Smith **or** _n = Jones

Condition Box (Cont.)

- QBE supports an interesting syntax for expressing alternative values

branch	*branch-name*	*branch-city*	*assets*
	P.	_x	

conditions
_x = (Brooklyn **or** Queens)

Condition Box (Cont.)

- Find all account numbers with a balance between $1,300 and $1,500

account	*account-number*	*branch-name*	*balance*
	P.		_x

conditions
_x ≥ 1300 _x ≤ 1500

- Find all account numbers with a balance between $1,300 and $2,000 but not exactly $1,500.

The Result Relation

- Find the *customer-name*, *account-number,* and *balance* for alll customers who have an account at the Perryridge branch.
  - We need to:
    - Join *depositor* and *account.*
    - Project *customer-name, account-number* and *balance.*
  - To accomplish this we:
    - Create a skeleton table, called *result,* with attributes *customer-name, account-number,* and *balance.*
    - Write the query.

The Result Relation (Cont.)

- The resulting query is:

account	*account-number*	*branch-name*	*balance*
	_y	Perryridge	_z

depositor	*customer-name*	*account-number*
	_x	_y

Ordering the Display of Tuples

- AO = ascending order, DO = descending order.
- E.g. list in ascending alphabetical order all customers who have an account at the bank

depositor	*customer-name*	*account-number*
	P.AO.	

- When sorting on multiple attributes, the sorting order is specified by including with each sort operator (AO or DO) an integer surrounded by parentheses.
- E.g. List all account numbers at the Perryridge branch in ascending alphabetic order with their respective account balances in descending order.

account	*account-number*	*branch-name*	*balance*
	P.AO(1).	Perryridge	P.DO(2).

Aggregate Operations

- The aggregate operators are AVG, MAX, MIN, SUM, and CNT
- The above operators must be postfixed with "ALL" (e.g.,

Aggregate Operations (Cont.)

- UNQ is used to specify that we want to eliminate duplicates
- Find the total number of customers having an account at the bank.

depositor	*customer-name*	*account-number*
	P.CNT.UNQ.	

Modification of the Database – Deletion

- Deletion of tuples from a relation is expressed by use of a D. command. In the case where we delete information in only some of the columns, null values, specified by –, are inserted.
- Delete customer Smith

customer	*customer-name*	*customer-street*	*customer-city*
D.	Smith		

- Delete the *branch-city* value of the branch whose name is "Perryridge".

branch	*branch-name*	*branch-city*	*assets*
	Perryridge	D.	

Deletion Query Examples

- Delete all loans with a loan amount between $1300 and $1500.
  - For consistency, we have to delete information from loan and borrower tables

loan	*loan-number*	*branch-name*	*amount*
D.	$-y$		$-x$

Modification of the Database – Insertion

- Insertion is done by placing the I. operator in the query expression.
- Insert the fact that account A-9732 at the Perryridge branch has a balance of $700.

account	*account-number*	*branch-name*	*balance*
I.	A-9732	Perryridge	700

Modification of the Database – Updates

Use the U. operator to change a value in a tuple without changing *all* values in the tuple. QBE does not allow users to update the primary key fields.

Update the asset value of the Perryridge branch to $10,000,000.

RELATIONAL DATABASE DEGIN

Data base design is a process in which you create a logical data model for a database, which store data of a company. It is performed after initial database study phase in the database life cycle. You use normalization technique to create the logical data model for a database and eliminate data redundancy. Normalization also allows you to organize data efficiently in a data base and reduce anomalies during data operation. Various normal forms, such as first, second and third can be applied to create a logical data model for a database. The second and third normal forms are based on partial dependency and transitivity dependency. Partial dependency occurs when a row of table is uniquely identified by one column that is a part of a primary key. A transitivity dependency ours when a non key column is uniquely identified by values in another non-key column of a table.

Data base design process:

We can identify six main phases of the database design process:

1. Requirement collection and analysis
2. Conceptual data base design
3. Choice of a DBMS
4. Data model mapping(logical database design)
5. physical data base design
6. database system implementation and tuning

1. **Requirement collection and analysis**

   Before we can effectively design a data base we must know and analyze the expectation of the users and the intended uses of the database in as much as detail.

2. **Conceptual data base design**

   The goal for this phase I s to produce a conceptual schema for the database that is independent of a specific DBMS.

   - We often use a high level data model such er-model during this phase
   - We specify as many of known database application on transactions as possible using a notation the is independent of any specific dbms.
   - Often the dbms choice is already made for the organization the intent of conceptual design still to keep , it as free as possible from implementation consideration.

3. **Choice of a DBMS**

   The choice of dbms is governed by a no. of factors some technical other economic and still other concerned with the politics of the organization.

   The economics and organizational factors that offer the choice of the dbms are:

   Software cost, maintenance cost, hardware cost, database creation and conversion cost, personnel cost, training cost, operating cost.

4. **Data model mapping (logical database design)**
   During this phase, we map the conceptual schema from the high level data model used on phase 2 into a data model of the choice dbms.
5. **Physical databse design**
   During this phase we design the specification for the database in terms of physical storage structure ,record placement and indexes.
6. **Database system implementation and tuning**
   During this phase, the database and application programs are implemented, tested and eventually deployed for service.

FUNCTIONAL DEPENDENCIES:

The functional dependency x→y

Holds on scema R if, in any legal relation r(R), for all pairs of tuples t1 and t2 in r such that t1[x]=t2[x]. it is also the case that t1[y]=t2[y]

For example, given the value of item code, there is only one value of item name for it. Thus item name is functionally dependent on item code. This is shows as:

Item code → item name

Similarly in table 1, given an order number, the date of the order is known. Thus : order no. Order date

Functional dependency may also be based on a composite attribute. For example, if we write

$$X, Z \rightarrow Y$$

it means that there is only one value of Y corresponding to given values of X, Z. In other words, Y is functionally dependent on the composite X, Z. In other words, Y is functionally dependent on the composite X, Z. In table 1 mentioned below, for example, Order no., and Item code together determine Qty. and Price. Thus :

Order no., Item code → Qty., Price

As another example, consider the relation

Student (**Roll no.**, Name, Address, Dept., Year of study)

Order no.	*Order date*	*Item code*	Quantity	*Price/unit*
1456	260289	3687	52	50.40
1456	260289	4627	38	60.20
1456	260289	3214	20	17.50
1886	040389	4629	45	20.25
1886	040389	4627	30	60.20
1788	040489	4627	40	60.20

Table 1: Normalized Form of the Relation

In this relation, Name is functionally dependent on Roll no. In fact, given the value of Roll no., the values of all the other attributes can be uniquely determined. Name and Department are not functionally dependent because given the name of a student, one cannot find his department uniquely. This is due to the fact that there may be more than one student with the same name. Name in this case is not a key. Department and Year of study are not functionally dependent as Year of study pertains to a student whereas Department is an independent attribute. The functional dependency in this relation is shown in the following figure as a dependency diagram. Such dependency diagrams shown in figure 1 are very useful in normalization.

Relation Key: Consider the relation of table 1. Given the Vendor code, the Vendor name and Address are uniquely determined. Thus Vendor code is the relation key. Given a relation, if the value of an attribute X uniquely determines the values of all other attributes in

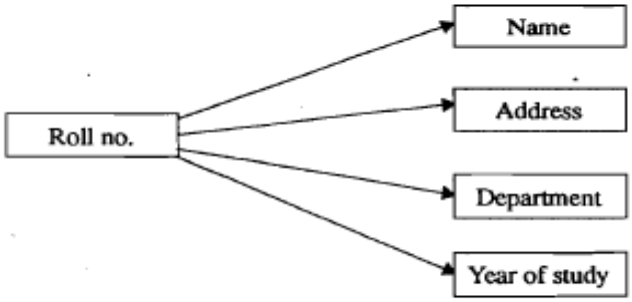


Figure 1: Dependency diagram for the relation "Student"

a row, then **X** is said to be the key of that relation. Sometimes more than one attribute is needed to uniquely determine other attributes in a relation row. In that case such a set of attributes is the key. In table 1, Order no. and Item code together form the key. In the relation "Supplies" (Vendor code, Item code, Qty. supplied, Date of supply, Price/unit), Vendor code and Item code together form the key. This dependency is shown in the following diagram (figure 2).

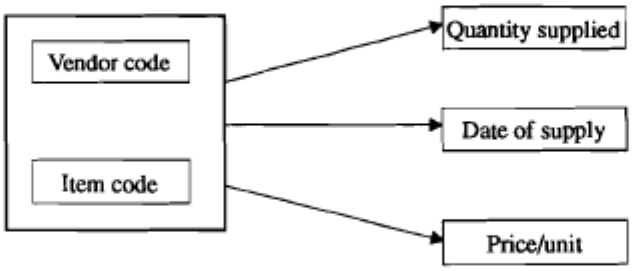


Figure 2: Dependency diagram for the relation "Supplies"

Observe that in the figure the fact that Vendor code and Item code together form a composite key is clearly shown by enclosing them together in a rectangle.

ANOMALIES IN A DATABASE

Consider the following relation scheme pertaining to the information about a student maintained by a university:

STDINF(Name, Course, Phone_No, Major, Prof, Grade)

Table 2 shows some tuples of a relation on the relation scheme **STDINF(Name, Course,** Phone_No, Major, Prof, Grade). The functional dependencies among its attributes are shown in Figure 3. The key of the relation is Name Course and the relation has, in addition, the following functional dependencies {Name → Phone_No, Name → Major, Name Course → Grade, Course → Prof}.

Name	Course	Phone_No	Major	Prof	Grade
Jones	353	237-4539	Comp Sci	Smith	A
Ng	329	427-7390	Chemistry	Turner	B
Jones	328	237-4539	Comp Sci	Clark	B
Martin	456	388-5183	Physics	James	A
Dulles	293	371-6259	Decision Sci	Cook	C
Duke	491	823-7293	Mathematics	Lamb	B
Duke	356	823-7293	Mathematics	Bond	in prog
Jones	492	237-4539	Comp Sci	Cross	in prog
Baxter	379	839-0827	English	Broes	C

Table 2: Student Data Representation in Relation STDINF

Here the attribute Phone_No, which is not in any key of the relation scheme STDINF, is not functionally dependent on the whole key but only one part of the key, namely, the attribute Name. Similarly, the attributes Major and Prof, which are not in any key of the relation scheme **STDINF** either, are fully functionally dependent on the attributes Name and Course, respectively. Thus the determinants of these functional dependencies are again not the entire key but only part of the key of the relation. Only the attribute Grade is fully functionally dependent on the key **Name Course.**

The relation scheme **STDINF** can lead to several undesirable problems:

- **Redundancy:** The aim of the database system is to reduce redundancy, meaning that information is to be stored only once. Storing information several times leads to the waste of storage space and an increase in the total size of the data stored.

Updates to the database with such redundancies have the potential of becoming inconsistent, as explained below. In the relation of table 2, the Major and Phone_No. of a student are stored several times in the database: once for each course that is or was taken by a student.

- **Update Anomalies:** Multiple copies of the same fact may lead to update anomalies or inconsistencies when an update is made and only some of the multiple copies are updated. Thus, a change in the Phone_No. of Jones must be made, for consistency, in all tuples pertaining to the student Jones. If one of the three tuples of Figure 3 is not changed to reflect the new Phone_No. of Jones, there will be an inconsistency in the data.

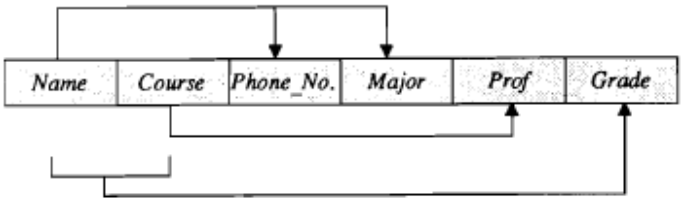


Figure 3: Function dependencies in STDINF

- **Insertion Anomalies:** If this is the only relation in the database showing the association between a faculty member and the course he or she teaches, the fact that a given processor is teaching a given course cannot be entered in the database unless a student is registered in the course. Also, if another relation also establishes a relationship between a course and a professor who teaches that course the information stored in these relations has to be consistent.
- **Deletion Anomalies:** If the only student registered in a given course discontinues the course, the information as to which professor is offering the course will be lost if this is the only relation in the database showing the association between a faculty member and the course she or he teaches. If another relation in the database also establishes the relationship between a course and a professor who teaches that course, the deletion of the last tuple in STDINF for a given course will not cause the information about the course's teacher to be lost.

The problems of database inconsistency and redundancy of data are similar to the problems that exist in the hierarchical and network models. These problems are addressed in the network model by the introduction of virtual fields and in the hierarchical model by the introduction of virtual records. In the relational model, the above problems can be remedied by decomposition. We define decomposition as follows:

Definition: Decomposition

The decomposition of a relation scheme $R = (A_1, A_2, \ldots A_n)$is its replacement by a set of relation schemes $\{R_1, R_2, \ldots R_m\}$, such that $R_1 \leq R$ for $1 \leq i \leq m$ and $R_1 \cup R_2 \cup R_m = R$.

A relation scheme R can be decomposed into a collection of relation schemes $\{R_1, R_2, R_3 \ldots R_m\}$ to eliminate some of the anomalies contained in the original relation R. Here the relation schemes R_i $(1 \leq i \leq m)$ are subsets of R and the intersection of $R_i \cap R_j$ for $i \neq j$ need not be empty. Furthermore, the union of R_j $(1 \leq i \leq m)$ is equal to R, i.e. $R = R_1 R_2 \ldots R_m$.

The problems in the relation scheme STDINF can be resolved if we replace it with the following relation schemes:

STUDENT_INFO (Name, Phone_No, Major)

TRANSCRIPT (Name, Course, Grade)

TEACHER (Course. Prof)

The first relation scheme gives the phone number and the major of each student and such information will be stored only once for each student. Any change in the phone number will thus require a change in only one tuple of this relation.

CLOSURE OF A SET OF FUNCTIONAL DEPEDENCIES

Given a relational schema R, a functional dependencies f on R is logically implied by a set of functional dependencies F on R if every relation instance r(R) that satisfies F also satisfies f.

The closure of F, denoted by F^+, is the set of all functional dependencies logically implied by F.
The closure of F can be found by using a collection of rules called **Armstrong axioms.**
Reflexivity rule: If A *is a set of attributes and B is subset or equal to A, then A*→B holds.
Augmentation rule: If A→B holds and C is a set of attributes, then CA→CB holds
Transitivity rule: If A→B holds and B→C holds, then A→C holds.
Union rule: If A→B holds and A→C then A→BC holds
Decomposition rule: If A→BC holds, then A→B holds and A→C holds.
Pseudo transitivity rule: If A→B holds and BC→D holds, then AC→D holds.

Suppose we are given a relation schema R=(A,B,C,G,H,I) and the set of function dependencies
A→B,A→C,CG→H,CG→I,B→H
We list several members of F^+ here:

- A→H, since A→B and B→H hold, we apply the transitivity rule.
- CG→HI. Since CG→H and CG→I , the union rule implies that CG→HI
- AG→I, since A→C and CG→I, the pseudo transitivity rule implies that AG→I holds

Algorithm of compute F+ :
To compute the closure of a set of functional dependencies F:

```
F+ = F
repeat
    for each functional dependency f in F+
        apply reflexivity and augmentation rules on f
        add the resulting functional dependencies to F+
    for each pair of functional dependencies f1and f2 in F+
        if f1 and f2 can be combined using transitivity
            then add the resulting functional dependency to F+
until F+ does not change any further
```

Closure of Attribute sets:-

To test whether a set α is a super key, we must devise an algorithm for omputing the set of attributes functionally determined by alpha. One way of doing this is to compute F+ take all functional dependencies. However doing so can be expensive, since F+ can be large.

Prepared by: Dr. Subhendu Kumar Rath

Closure of Attribute Sets

Given a set of attributes α, define the *closure* of α under *F* (denoted by α^+) as the set of attributes that are functionally determined by α under *F:* $\alpha \rightarrow \beta$ is in F^+ <=> $\beta \subseteq \alpha^+$

Algorithm to compute α^+, the closure of α under *F* *result* := α; **while** (changes to *result*) **do** **for each** $\beta \rightarrow \gamma$ **in** *F* **do**

begin **if** $\beta \subseteq$ *result* **then** *result* := *result* $\cup \gamma$ **end**

Prepared by: Dr. Subhendu Kumar Rath

Example of Attribute Set Closure

$R = (A, B, C, G, H, I)$

$F = \{A \rightarrow B \quad A \rightarrow C \quad CG \rightarrow H \quad CG \rightarrow I \quad B \rightarrow H\}$

$(AG)^+$

1. *result* = AG
2. *result* = $ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
3. *result* = $ABCGH$ $(CG \rightarrow H$ and $CG \subseteq AGBC)$
4. *result* = $ABCGHI$ $(CG \rightarrow I$ and $CG \subseteq AGBCH)$

Is AG a candidate key?

Is AG a super key?

Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$

Is any subset of AG a superkey?

Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$

Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$

Prepared by: Dr. Subhendu Kumar Rath

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

Testing for superkey:

To test if α is a superkey, we compute α^+, and check if α^+ contains all attributes of *R*.

Testing functional dependencies

To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^+$.

That is, we compute α^+ by using attribute closure, and then check if it contains β.

Is a simple and cheap test, and very useful

Computing closure of F

For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

Prepared by: Dr. Subhendu Kumar Rath

Canonical Cover

Sets of functional dependencies may have redundant dependencies that can be inferred from the others

Eg: $A \rightarrow C$ is redundant in: $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow C\}$

Parts of a functional dependency may be redundant

E.g. on RHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow CD\}$ can be simplified to $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$

E.g. on LHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad AC \rightarrow D\}$ can be simplified to $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$

Intuitively, a canonical cover of F is a “minimal” set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies

Extraneous Attributes

Consider a set *F* of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in *F*.

Attribute A is extraneous in α if $A \in \alpha$ and *F* logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.

Attribute *A* is extraneous in β if $A \in \beta$ and the set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies *F.*

Note: implication in the opposite direction is trivial in each of the cases above, since a "stronger" functional dependency always implies a weaker one

Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$

B is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (I.e. the result of dropping *B* from $AB \rightarrow C$).

Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$

C is extraneous in $AB \rightarrow CD$ since AB $\rightarrow C$ can be inferred even after deleting *C*

Testing if an Attribute is Extraneous

Consider a set *F* of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in *F*.

To test if attribute $A \in \alpha$ is extraneous in α

compute $(\{\alpha\} - A)^+$ using the dependencies in *F*

check that $(\{\alpha\} - A)^+$ contains A; if it does, *A* is extraneous

To test if attribute $A \in \beta$ is extraneous in β

compute α^+ using only the dependencies in $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$,

check that α^+ contains *A;* if it does, *A* is extraneous

Prepared by: Dr. Subhendu Kumar Rath

Canonical Cover

A *canonical cover* for *F* is a set of dependencies F_c such that

- *F* logically implies all dependencies in F_c, and
- F_c logically implies all dependencies in *F*, and
- No functional dependency in F_c contains an extraneous attribute, and
- Each left side of functional dependency in F_c is unique.

To compute a canonical cover for *F*: **repeat** Use the union rule to replace any dependencies in *F* $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$ Find a functional dependency $\alpha \rightarrow \beta$ with an extraneous attribute either in α or in β If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$ **until** *F* does not change

Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

Prepared by: Dr. Subhendu Kumar Rath

Example of Computing a Canonical Cover

$R = (A, B, C)$ $F = \{A \rightarrow BC \quad B \rightarrow C \quad A \rightarrow B \quad AB \rightarrow C\}$

Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$

Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$

A is extraneous in $AB \rightarrow C$

Check if the result of deleting A from $AB \rightarrow C$ is implied by the other dependencies

Yes: in fact, $B \rightarrow C$ is already present!

Set is now $\{A \rightarrow BC, B \rightarrow C\}$

C is extraneous in $A \rightarrow BC$

Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies

Yes: using transitivity on $A \rightarrow B$ *and* $B \rightarrow C$.

Can use attribute closure of A in more complex cases

The canonical cover is: $A \rightarrow B \qquad B \rightarrow C$

LOSS LESS DECOMPOSITION:

A decomposition of a relation scheme R<S,F> into the relation schemes Ri(1<=i<=n) is said to be a lossless join decomposition or simply lossless if for every relation R that satisfies the FDs in F, the natural join of the projections or R gives the original relation R, i.e,

$$R=\prod_{R1}(R) \bowtie \prod_{R2}(R) \bowtie \ldots\ldots\ldots \bowtie \prod_{Rn}(R)$$

If R is subset of $\prod_{R1}(R) \bowtie \prod_{R2}(R) \bowtie \ldots\ldots\ldots \bowtie \prod_{Rn}(R)$
Then the decomposition is called lossy.

DEPEDENCY PRSERVATION:

Given a relation scheme R<S,F> where F is the associated set of functional dependencies on the attributes in S,R is decomposed into the relation schemes R1,R2,…Rn with the fds F1,F2…Fn, then this decomposition of R is dependency preserving if the closure of F' (where F'=F1 U F2 U … Fn)

Example:
Let R(A,B,C) AND F={A→B}. Then the decomposition of R into R1(A,B) and R2(A,C) is lossless because the FD { A→B} is contained in R1 and the common attribute A is a key of R1.

Example:

Let R(A,B,C) AND F={A→B}. Then the decomposition of R into R1(A,B) and R2(B,C) is not lossless because the common attribute B does not functionally determine either A or C. i.e, it is not a key of R1 or R 2.

Example:

Let R(A,B,C,D) and F={A→B, A→C, C→D,}. Then the decomposition of R into R1(A,B,C) with the FD F1={ A→B , A→C }and R2(C,D) with FD F2={ C→D} . In this decomposition all the original FDs can be logically derived from F1 and F2, hence the decomposition is dependency preserving also . the common attribute C forms a key of R2. The decomposition is lossless.

Example:

Let R(A,B,C,D) and F={A→B, A→C, A→D,}. Then the decomposition of R into R1(A,B,D) with the FD F1={ A→B , A→D }and R2(B,C) with FD F2={ } is lossy because the common attribute B is not a candidate key of either R1 and R2 .
In addition , the fds A→C is not implied by any fds R1 or R2. Thus the decomposition is not dependency preserving.

Full functional dependency:
Given a relational scheme R and an FD X→Y ,Y is fully functional dependent on X if there is no Z, where Z is a proper subset of X such that Z→Y. The dependency X→Y is left reduced, there being no extraneous attributes attributes in the left hand side of the dependency.

Partial dependency:
Given a relation dependencies F defined on the attributes of R and K as a candidate key ,if X is a proper subset of K and if F|= X→A, then A is said to be partial dependent on K

Prime attribute and non prime attribute:
A attribute A in a relation scheme R is a **prime attribute** or simply **prime** if A is part of any candidate key of the relation. If A is not a part of any candidate key of R, A is called a nonprime attribute or simply **non prime** .

Trivial functional dependency:
A FD X→Y is said to be a trivial functional dependency if Y is subset of X.

NORMALIZATION

The basic objective of normalization is to reduce redundancy which means that information is to be stored only once. Storing information several times leads to wastage of storage space and increase in the total size of the data stored. Relations are normalized so that when relations in a database are to be altered during the life time of the database, we do not lose information or introduce inconsistencies. The type of alterations normally needed for relations are:

- Insertion of new data values to a relation. This should be possible without being forced to leave blank fields for some attributes.
- Deletion of a tuple, namely, a row of a relation. This should be possible without losing vital information unknowingly.
- Updating or changing a value of an attribute in a tuple. This should be possible without exhaustively searching all the tuples in the relation.

PROPERTIES OF NORMALIZED RELATIONS

Ideal relations after normalization should have the following properties so that the problems mentioned above do not occur for relations in the (ideal) normalized form:

1. No data value should be duplicated in different rows unnecessarily.
2. A value must be specified (and required) for every attribute in a row.
3. Each relation should be self-contained. In other words, if a row from a relation is deleted, important information should not be accidentally lost.
4. When a mw is added to a relation, other relations in the database should not be affected.
5. A value of an attribute in a tuple may be changed independent of other tuples in the relation and other relations.

The idea of normalizing relations to higher and higher normal forms is to attain the goals of having a set of ideal relations meeting the above criteria.

Unnormalized relation:

Defn: An unnormalized relation contains non atomic values.

Each row may contain multiple set of values for some of the columns, these multiple values in a single row are also called non atomic values.

Order no.	*Order date*	*Item code*	Quantity	*Price/unit*
1456	260289	3687	52	50.40
		4627	38	60.20
		3214	20	17.50
1886	040389	4629	45	20.25
		4627	30	60.20
1788	040489	4627	40	60.20

FIRST NORMAL FORM:

Defn: A relation scheme is said to be in first normal form(1NF) if the values in the domain of each attribute of the relation are atomic. In other words, only one value is associated with each attribute and the value is not a set of values or a list of values.

Order no.	*Order date*	*Item code*	Quantity	*Price/unit*
1456	260289	3687	52	50.40
1456	260289	4627	38	60.20
1456	260289	3214	20	17.50
1886	040389	4629	45	20.25
1886	040389	4627	30	60.20
1788	040489	4627	40	60.20

Functional dependencies are:

item code → item name

orderno → orderdate

Order no., Item code → Qty., Price

SECOND NORMAL FORM:

Defn: A relation scheme R<S,F> is in second normal form(2NF) if it is in the !NF and if all non prime attributes are fully functionally dependent on the relation keys.

A relation is said to be in**2NF** if it is in **1NF** and non-key attributes are functionally dependent on the key attribute(s). Further. if the key has more than one attribute then no non-key attributes should be functionally dependent upon a part of the key attributes. Consider, for example, the relation given in table 1. This relation is in **1NF.** The key is (Order no.. Item code). The dependency diagram for attributes of this relation is shown in figure 5. The non-key attribute Price_Unit is functionally dependent on Item code which is part of the relation key. Also, the non-key attribute Order date is functionally dependent on Order no. which is a part of the relation key.
Thus the relation is not in **2NF.** It can be transformed to 2NF by splitting it into three relations as shown in table 3.
In table 3 the relation Orders has Order **no.** as the key. The relation Order details has the composite key Order **no.** and Item code. In both relations the non-key attributes are functionally dependent on the whole key. Observe that by transforming to 2NF relations the

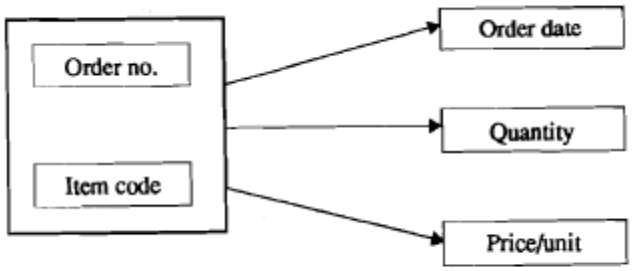


Figure 5: Dependency diagram for the relation given in table

repetition of Order date (table 1) has been removed. Further, if an order for an item is cancelled, the price of an item is not lost. For example, if Order no. **1886** for Item code **4629** is cancelled in table 1, then the fourth row will be removed and the price of the item is lost. In table 3 only the fourth row of the table 3(b) is omitted. The item price is not lost as it is available in table 3(c). The data of the order is also not lost as it is in table 3(a).

(a) Orders

Order no	*Order date*
1456	260289
1886	040389
1788	040489

(b) Order Details

Order no.	Item Code	Qty.
1456	3687	52
1456	4627	38
1456	3214	20
1886	4629	45
1886	4627	30
1788	4627	40

(c) Prices

Item code	Price/ unit
3687	50.40
4627	60.20
3214	17.50
4629	20.25

Table 3: Splitting of Relation given in table 1 into 2NF Relations

These relations in 2NF form meet all the "ideal" conditions specified. Observe that the three relations obtained as self-contained. There is no duplication of data within a relation.

THIRD NORMAL FORM:

Defn: A relational scheme R<S,F> is in third normal form(3NF) if for all non trivial function dependencies in F+ of the form X→A, either X contains a key(i.e, X is super key) or A is a prime key attribute.

A Third Normal Form normalization will be needed where all attributes in a relation tuple are not functionally dependent only on the key attribute. If two non-key attributes are functionally dependent, then there will be unnecessary duplication of data. Consider the relation given in table 4. Here. Roll no. is the key and all other attributes are

Roll no.	Name	Department	Year	Hostel name
1784	Raman	Physics	1	Ganga
1648	Krishnan	Chemistry	1	Ganga
1768	Gopalan	Mathematics	2	Kaveri
1848	Raja	Botany	2	Kaveri
1682	Maya	Geology	3	Krishna
1485	Singh	Zoology	4	Godavari

Table 4: A 2NF Form Relation

functionally dependent on it. Thus it is in 2NF. If it is known that in the college all first year students are accommodated in Ganga hostel, all second year students in Kaveri, all third year students in Krishna, and all fourth year students in Godavari, then the non-key attribute Hostel name is dependent on the non-key attribute Year. This dependency is shown in figure 6.

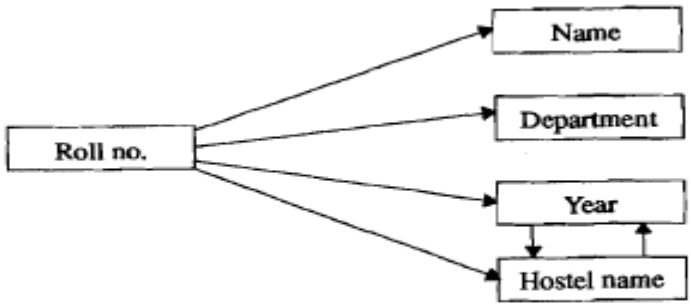


Figure 6: Dependency diagram for the relation

Observe that given the year of student, his hostel is known and vice versa. The dependency of hostel on year leads to duplication of data as is evident from table 4. If it is decided to ask all first year students to move to Kaveri hostel, and all second year students to Ganga hostel. this change should be made in many places in table 4. Also, when a student's year of study changes, his hostel change should also be noted in Table 4. This is undesirable. Table 4 is said to be in **3NF** if it is in **2NF** and no non-key attribute is functionally dependent on any other non-key attribute. Table 4 is thus not in **3NF.** To transform it to **3NF,** we should introduce another relation which includes the functionally related non-key attributes. This is shown in table 5.

Roll no.	Name	Department	Year
1784	Raman	Physics	1
1648	Krishnan	Chemistry	1
1768	Gopalan	Mathematics	2
1848	Raja	Botany	2
1682	Maya	Geology	3
1485	Singh	Zoology	4

Year	Hostel name
1	Ganga
2	Kaveri
3	Krishna
4	Godavari

Table 5: Conversion of table 4 into two 3NF relations

BOYCE CODD NORMAL FORM:

Defn: a normalized relation scheme R<S,F> is in Boyce Codd normal form if for every nontrivial FD in F+ of the form X→A where X is subset of S and A∈S, X is a super key of R.

Assume that a relation has more than one possible key. Assume further that the composite keys have a common attribute. If an attribute of a composite key is dependent on an attribute of the other composite key, a normalization called BCNF is needed. Consider. as an

example, the relation Professor:

Professor (**Professor code,** Dept., Head of Dept., Parent time)

It is assumed that
1. A professor can work in more than one department
2. The percentage of the time he spends in each department is given.
3. Each department has only one Head of Department.

The relationship diagram for the above relation is given in figure 8. Table 6 gives the relation attributes. The two possible composite keys are professor code and Dept. or Professor code and Hcad of Dept. Observe that department as well as Head of Dept. are not non-key attributes. They are a part of a composite key

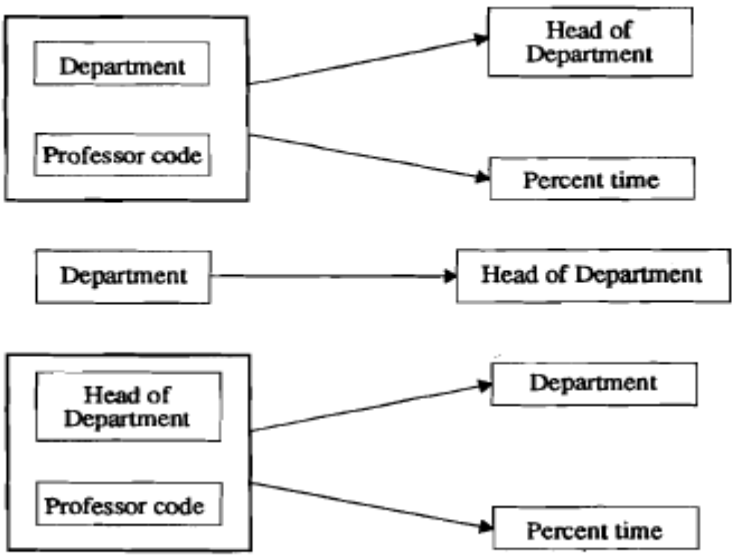


Figure 8: Dependency diagram of Professor relation

Professor Code	Department	Head of Dept.	Parent
P1	Physics	Ghosh	50
P1	Mathematics	Krishnan	50
P2	Chemistry	Rao	25
P2	Physics	Ghosh	75
P3	Mathematics	Krishnan	100

Table 6: Normalization of Relation "Professor"

The relation given in table 6 is in 3NF. Observe, however, that the names of Dept. and Head of Dept. are duplicated. Further, if Professor P2 resigns, rows 3 and 4 are deleted. We lose the information that Rao is the Head of Department of Chemistry.

The normalization of the relation is done by creating a new relation for Dept. and Head of Dept. and deleting Head of Dept. from Professor relation. The normalized relations are shown in the following table 7.

(a)

Professor code	Department	Percent time
P1	Physics	50
P1	Mathematics	50
P2	Chemistry	25
P2	Physics	75
P3	Mathematics	100

(b)

Department	Head of Dept.
Physics	Ghosh
Mathematics	Krishnan
Chemistry	Rao

Table 7: Normalized Professor Relation in BCNF

and the dependency diagrams for these new relations in figure 8. The dependency diagram gives the important clue to this normalization step as is clear from figures 8 and 9.

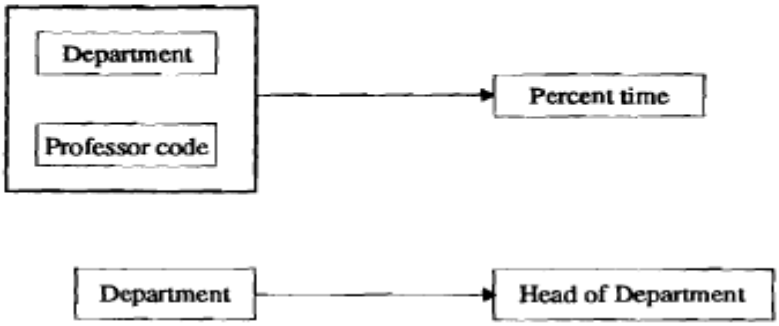


Figure 9: Dependency diagram of Professor relation

MULTIVALUED DEPEDENCY:

*Defn:*Given a relation scheme R, Le X and Y be subsets of attributes of R. then the multi valued dependency X →→Y holds in a relation R defined on R if given two tuples t1 and t2 in R with t1(X)=t2(X);

R contains two tuples t3 and t4 with the following characteristics: t1,t2,t3,t4 have the X value i.e,

T1(X)= T2(X)=T3(X)= T4(X)

The Y values of t1 and t3 are the same and the Y values of t2 and t4 are the same .i.e,

T1(Y)= T2(Y)=T3(Y)= T4(Y)

TRIVIAL MULTIVALED DEPEDENCY:

A trivial multi valued dependency is one that is satisfied by all relations R on a relation scheme R with XY is subset or equal to R. Thus, a MVD X →→Y is trivial if Y is sub set or equal to X or XY=R.

FOURTH NORMAL FORM:

Defn:
Given a relation scheme R such that the set D of FDs and MVDs are satisfied, consider a set attributes X and Y where X is subset or equal to R,Y is subset or equal to Y. The reltion scheme R is in 4NF if for all mutivalued dependencies of the form X →→Y Є D+ Either X →→Y is a trivial MVD or X is super key of R.

When attributes in a relation have multivalucd dependency, further Normalisation to 4NF and 5NF are required. We will illustrate this with an example. Consider a vendor supplying
many items to many projects in an organisation. The following are the assumptions:
1. A vendor is capable of supplying many items.
2. A project uses many items
3. A vendor supplies to many projects.
4. An item may be supplied by many vendors.
Table 8 gives a relation for this problem and figure 10 lhe dependency diagram(s).

Vendor code	Item code	Project no.l
V1	I1	P1
V1	I2	P1
V1	I1	P3
V1	I2	P3
V2	I2	P1
V2	I3	P1
V3	I1	P2
V3	I1	P2

Table 8: Vendor-supply-projects Relation

The relation given in table 8 has a number of problems. For example:

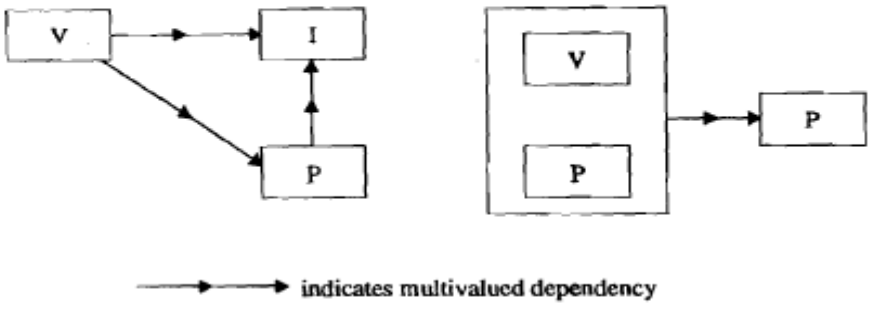


Figure 10 : Dependency diagrams of vendor-supply-project relation

- If vendor V1 has to supply to project P2, but the item is not yet decided, then a row with a blank for item code has to be introduced.
- The information about item 1 is stored twice for vendor V3.

Observe that the relation given in Table 8 is in 3NF and also in BCNF. It still has the problems mentioned above. The problem is reduced by expressing this relation as two relations in the Fourth Normal Form (4NF). A relation is in 4NF if it has no more than one independent multivalued dependency or one independent multivalued dependency with a functional dependency.

Table 8 can be expressed as the two 4NF relations given in Table 9. The fact that vendors are capable of supplying certain items and that they are assigned to supply for some projects in independently specified in the 4NF relation.

(a) Vendor Supply	
Vendor code	Item code
V1	I1
V1	I2
V2	I2
V2	I3
V3	I1

(b) Vendor project	
Vendor code	Project no.
V1	P1
V1	P3
V2	P1
V3	P1
V3	P2

Table 9: Vendor-supply-project Relations in 4NF

These relations still have a problem. Even though vendor V1's capability to supply items and his allotment to supply for specified projects may not need it. We thus need another relation which specifies this. This is called 5NF form. The 5NF relations are the relations in Table 9(a) and 9(b) together with the relation given in table 10.

Project no.	Item code
P1	I1
P1	I2
P2	I1
P3	I1
P3	I3

Table 10: 5NF Additional Relation

In table 11 we summarise the normalisation steps already explained.

Input relation	Transformation	Output relation
All relations	Eliminate variable length records. Remove multiattribute lines in table	1NF
1NF relation	Remove dependency of non-key attribute on part of a multiattribute key	2NF
2NF	Remove dependency of non-key attributes on other non-key attributes	3NF
3NF	Remove dependency of an attribute of a multiattribute key on an attribute of another (overlapping) multi-attribute key	BCNF
BCNF	Remove more than one independent multivalued dependency from relation by splitting relation	4 NF
4NF	Add one relation relating attributes with multivalued dependency to the two relations with multivalued dependency	5 NF

Table 11: Summary of Normalisation Steps

TRANSCATION:

A **transaction** is a **unit** of program execution that accesses and possibly updates various data items. Usually, a transaction is initiated by a user program written in a high-level data-manipulation language or programming language (for example, SQL, COBOL, C, C++, or Java), where it is delimited by statements (or function calls) of the form **begin transaction** and **end transaction**. The transaction consists of all operations executed between the **begin transaction** and **end transaction**. To ensure integrity of the data, we require that the database system maintain the following properties of the transactions:

- **Atomicity**. Either all operations of the transaction are reflected properly in the database, or none are.
- **Consistency**. Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.
- **Isolation**. Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions *Ti* and *Tj* , it appears to *Ti* that either *Tj* finished execution before *Ti* started, or *Tj* started execution after *Ti* finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.
- **Durability**. After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

These properties are often called the **ACID properties**; the acronym is derived from the first letter of each of the four properties.
Let *Ti* be a transaction that transfers $50 from account *A* to account *B*. This transaction can be defined as
Ti: read(*A*);
A := *A* − 50;
write(*A*);

read(B);
$B := B + 50$;
write(B).

TRANSCATION STATE:

A transaction must be in one of the following states:
• **Active**, the initial state; the transaction stays in this state while it is executing
• **Partially committed**, after the final statement has been executed
• **Failed**, after the discovery that normal execution can no longer proceed
• **Aborted**, after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction
• **Committed**, after successful completion

The state diagram corresponding to a transaction appears in Figure 15.1. We say that a transaction has committed only if it has entered the committed state. Similarly,we say that a transaction has aborted only if it has entered the aborted state. A transaction is said to have **terminated** if has either committed or aborted.
A transaction starts in the active state. When it finishes its final statement, it enters the partially committed state. At this point, the transaction has completed its execution, but it is still possible that it may have to be aborted, since the actual output may still be temporarily residing in main memory, and thus a hardware failure may preclude its successful completion.
The database system then writes out enough information to disk that, even in the event of a failure, the updates performed by the transaction can be re-created when the system restarts after the failure. When the last of this information is written out, the transaction enters the committed state.

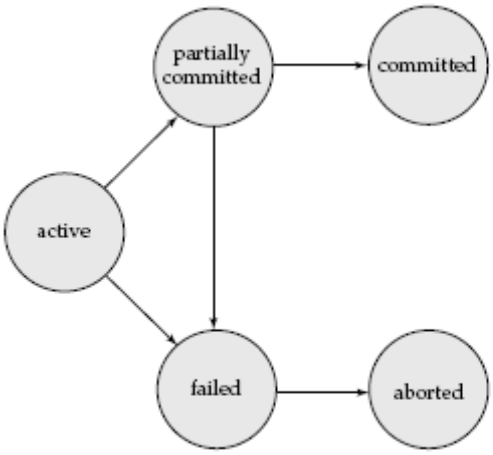


Figure 15.1 State diagram of a transaction.

- It can **restart** the transaction, but only if the transaction was aborted as a result of some hardware or software error that was not created through the internal logic of the transaction. A restarted transaction is considered to be a new transaction.

- It can **kill** the transaction. It usually does so because of some internal logical error that can be corrected only by rewriting the application program, or because the input was bad, or because the desired data were not found in the database.

Concurrent Executions:

Multiple transactions are allowed to run concurrently in the system.

Advantages are:

increased processor and disk utilization, leading to better transaction *throughput:* one transaction can be using the CPU while another is reading from or writing to the disk

reduced average response time for transactions: short transactions need not wait behind long ones.

Concurrency control schemes – mechanisms to achieve isolation, i.e., to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

Schedules

Schedules – sequences that indicate the chronological order in which instructions of concurrent transactions are executed

- ★ a schedule for a set of transactions must consist of all instructions of those transactions
- ★ must preserve the order in which the instructions appear in each individual transaction

Example Schedules

Let $T1$ transfer \$50 from A to B, and $T2$ transfer 10% of the balance from A to B. The following is a serial schedule (Schedule 1 in the text), in which $T1$ is followed by $T2$.

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

Let $T1$ and $T2$ be the transactions defined previously. The following schedule (Schedule 3 in the text) is not a serial schedule, but it is *equivalent* to Schedule 1.

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	

In both Schedule 1 and 3, the sum A + B is preserved.

Serializability:

- Basic Assumption – Each transaction preserves database consistency.
- Thus serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
  - conflict serializability
  - view serializability
- We ignore operations other than **read** and **write** instructions, and we assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes. Our simplified schedules consist of only **read** and **write** instructions.

Conflict Serializability

- Instructions *li* and *lj* of transactions *Ti* and *Tj* respectively, **conflict** if and only if there exists some item *Q* accessed by both *li* and *lj*, and at least one of these instructions wrote *Q*.
  - *li* = **read**(*Q*), *lj* = **read**(*Q*). *li* and *lj* don't conflict.
  - *li* = **read**(*Q*), *lj* = **write**(*Q*). They conflict.
  - *li* = **write**(*Q*), *lj* = **read**(*Q*). They conflict
  - *li* = **write**(*Q*), *lj* = **write**(*Q*). They conflict
- Intuitively, a conflict between *li* and *lj* forces a (logical) temporal order between them. If *li* and *lj* are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.
- If a schedule *S* can be transformed into a schedule *S´* by a series of swaps of non-conflicting instructions, we say that *S* and *S´* are **conflict equivalent**.
- We say that a schedule *S* is **conflict serializable** if it is conflict equivalent to a serial schedule
- Example of a schedule that is not conflict serializable:

*T*3	*T*4
read(*Q*)	
	write(*Q*)
write(*Q*)	

We are unable to swap instructions in the above schedule to obtain either the serial schedule < *T*3, *T*4 >, or the serial schedule < *T*4, *T*3 >.

- Schedule 3 below can be transformed into Schedule 1, a serial schedule where *T*2 follows *T*1, by series of swaps of non-conflicting instructions. Therefore Schedule 3 is conflict serializable.

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

View Serializability

Let *S* and *S´* be two schedules with the same set of transactions. *S* and *S´* are **view equivalent** if the following three conditions are met:

1. For each data item *Q*, if transaction *Ti* reads the initial value of *Q* in schedule *S*, then transaction *Ti* must, in schedule *S´*, also read the initial value of *Q*.
2. For each data item *Q* if transaction *Ti* executes **read**(*Q)* in schedule *S*, and that value was produced by transaction *Tj* (if any), then transaction *Ti* must in schedule *S´* also read the value of *Q* that was produced by transaction *Tj* .
3. For each data item *Q*, the transaction (if any) that performs the final **write**(*Q*) operation in schedule *S* must perform the final **write**(*Q*) operation in schedule *S´*.

As can be seen, view equivalence is also based purely on **reads** and **writes** alone.

- A schedule *S* is **view serializable** it is view equivalent to a serial schedule.
- Every conflict serializable schedule is also view serializable.
- Schedule 9 (from text) — a schedule which is view-serializable but *not* conflict serializable.

- Every view serializable schedule that is not conflict serializable has **blind writes.**

Cascading rollback – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)

If *T*10 fails, *T*11 and *T*12 must also be rolled back.
Can lead to the undoing of a significant amount of work

Concurrency Control

- Lock-Based Protocols
- Timestamp-Based Protocols
- Validation-Based Protocols
- Multiple Granularity
- Multiversion Schemes
- Deadlock Handling

Lock-Based Protocols

- A lock is a mechanism to control concurrent access to a data item
- Data items can be locked in two modes :
  - *exclusive (X) mode*. Data item can be both read as well as written. X-lock is requested using **lock-X** instruction.
  - *shared (S) mode*. Data item can only be read. S-lock is requested using **lock-S** instruction.
- Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.
- Lock-compatibility matrix

	S	X
S	true	false
X	false	false

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
- Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.
- Example of a transaction performing locking:

  T2: **lock-S***(A)*;
  read *(A)*;
  unlock*(A)*;
  lock-S*(B)*;
  read *(B)*;
  unlock*(B)*;
  display*(A+B)*

- Locking as above is not sufficient to guarantee serializability — if *A* and *B* get updated in-between the read of *A* and *B*, the displayed sum would be wrong.

- A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

Pitfalls of Lock-Based Protocols:

- Consider the partial schedule

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

- Neither *T3* nor *T4* can make progress — executing **lock-S***(B)* causes *T4* to wait for *T3* to release its lock on *B*, while executing **lock-X***(A)* causes *T3* to wait for *T4* to release its lock on *A*.
- Such a situation is called a **deadlock**.
    - To handle a deadlock one of *T3* or *T4* must be rolled back and its locks released.

The Two-Phase Locking Protocol

One protocol that ensures serializability is the **two-phase locking protocol**. This protocol requires that each transaction issue lock and unlock requests in two phases:

1. Growing phase. A transaction may obtain locks, but may not release any lock.

2. Shrinking phase. A transaction may release locks, but may not obtain any new locks.

Initially, a transaction is in the growing phase. The transaction acquires locks as needed. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests.

For example, transactions *T*3 and *T*4 are two phase. On the other hand, transactions *T*1 and *T*2 are not two phase. Note that the unlock instructions do not need to appear at the end of the transaction. For example, in the case of transaction *T*3, we could move the unlock(*B*) instruction to just after the lock-X(*A*) instruction, and still retain the two-phase locking property.

```
T1: lock-X(B);
    read(B);
    B := B − 50;
    write(B);
    unlock(B);
    lock-X(A);
    read(A);
    A := A + 50;
    write(A);
    unlock(A).
```

Figure 16.2 Transaction T_1.

```
T2: lock-S(A);
    read(A);
    unlock(A);
    lock-S(B);
    read(B);
    unlock(B);
    display(A + B).
```

Figure 16.3 Transaction T_2.

```
T3: lock-X(B);
    read(B);
    B := B − 50;
    write(B);
    lock-X(A);
    read(A);
    A := A + 50;
    write(A);
    unlock(B);
    unlock(A).
```

Figure 16.5 Transaction T_3.

```
T4: lock-S(A);
    read(A);
    lock-S(B);
    read(B);
    display(A + B);
    unlock(A);
    unlock(B).
```

Figure 16.6 Transaction T_4.

Cascading rollbacks can be avoided by a modification of two-phase locking called the **strict two-phase locking protocol**. This protocol requires not only that locking be two phase, but also that all exclusive-mode locks taken by a transaction be held until that transaction commits. This requirement ensures that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits, preventing any other transaction from reading the data.

Timestamp-Based Protocols

Timestamps:

With each transaction *Ti* in the system, we associate a unique fixed timestamp, denoted by TS(*Ti*). This timestamp is assigned by the database system before the transaction *Ti* starts execution. If a transaction *Ti* has been assigned timestamp TS(*Ti*), and a new

transaction *Tj* enters the system, then TS(*Ti*) < TS(*Tj*). There are two simple methods for implementing this scheme:

1. Use the value of the *system clock* as the timestamp; that is, a transaction's timestampis equal to the value of the clock when the transaction enters the system.

2. Use a **logical counter** that is incremented after a new timestamp has been assigned; that is, a transaction's timestamp is equal to the value of the counter When the transaction enters the system.

The timestamps of the transactions determine the serializability order. Thus, if TS(*Ti*) < TS(*Tj*), then the system must ensure that the produced schedule is equivalent to a serial schedule in which transaction *Ti* appears before transaction *Tj* .

To implement this scheme, we associate with each data item *Q* two timestamp values:

• **W-timestamp**(*Q*) denotes the largest timestamp of any transaction that executed write(*Q*) successfully.

• **R-timestamp**(*Q*) denotes the largest timestamp of any transaction that executed read(*Q*) successfully.

These timestamps are updated whenever a new read(*Q*) or write(*Q*) instruction is executed.

The Timestamp-Ordering Protocol:

The **timestamp-ordering protocol** ensures that any conflicting read and write operations are executed in timestamp order. This protocol operates as follows:

1. Suppose that transaction *Ti* issues read(*Q*).
   - If TS(*Ti*) < W-timestamp(*Q*), then *Ti* needs to read a value of *Q* that was already overwritten. Hence, the read operation is rejected, and *Ti* is rolled back.
   - If TS(*Ti*) ≥ W-timestamp(*Q*), then the read operation is executed, and Rtimestamp(*Q*) is set to the maximum of R-timestamp(*Q*) and TS(*Ti*).
2. Suppose that transaction *Ti* issues write(*Q*).
   - If TS(*Ti*) < R-timestamp(*Q*), then the value of *Q* that *Ti* is producing was needed previously, and the system assumed that that value would never be produced. Hence, the system rejects the write operation and rolls *Ti* back.
   - If TS(*Ti*) < W-timestamp(*Q*), then *Ti* is attempting to write an obsolete value of *Q*. Hence, the system rejects this write operation and rolls *Ti* back. Otherwise, the system executes the write operation and sets W-timestamp(*Q*) to TS(*Ti*).
   -

Deadlock Handling

System is deadlocked if there is a set of transactions such that every transaction in the set is waiting for another transaction in the set.

Deadlock prevention protocols ensure that the system will *never* enter into a deadlock state. Some prevention strategies :

- Require that each transaction locks all its data items before it begins execution (predeclaration).
- Impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by the partial order (graph-based protocol).

Deadlock Handling

Consider the following two transactions:

T_1: write (X) T_2: write(Y)
write(Y) write(X)

Schedule with deadlock

T	T
lock-X on X write (X)	
	lock-X on Y write (X) wait for **lock-X** on X
wait for **lock-X** on	

Following schemes use transaction timestamps for the sake of deadlock prevention alone.

1. The wait–die scheme is a non preemptive technique. When transaction Ti requests
a data item currently held by Tj , Ti is allowed to wait only if it has a timestamp smaller than that of Tj (that is, Ti is older than Tj). Otherwise, Ti is rolled back (dies).
For example, suppose that transactions T22, T23, and T24 have timestamps 5, 10, and 15, respectively. If T22 requests a data item held by T23, then T22 will wait. If T24 requests a data item held by T23, then T24 will be rolled back.

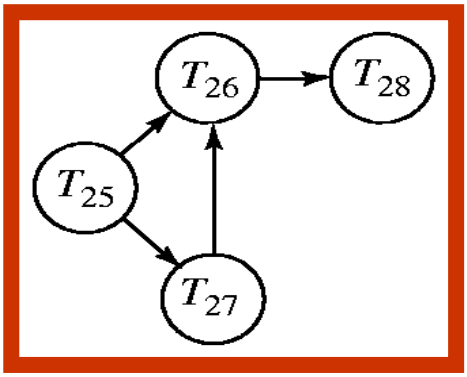
There are, however, significant differences in the way that the two schemes operate.

- In the wait–die scheme, an older transaction must wait for a younger one to release its data item. Thus, the older the transaction gets, the more it tends to wait. By contrast, in the wound–wait scheme, an older transaction never waits for a younger transaction.
- In the wait–die scheme, if a transaction Ti dies and is rolled back because it requested a data item held by transaction Tj, then Ti may reissue the same sequence of requests when it is restarted. If the data item is still held by Tj , then Ti will die again. Thus, Ti may die several times before acquiring the needed data item. Contrast this series of events with what happens in the wound–wait scheme. Transaction Ti is wounded and rolled back because Tj requested a data item that it holds. When Ti is restarted and requests the data item now being held by Tj , Ti waits. Thus, there may be fewer rollbacks in the wound–wait scheme.
  The major problem with both of these schemes is that unnecessary rollbacks may occur.

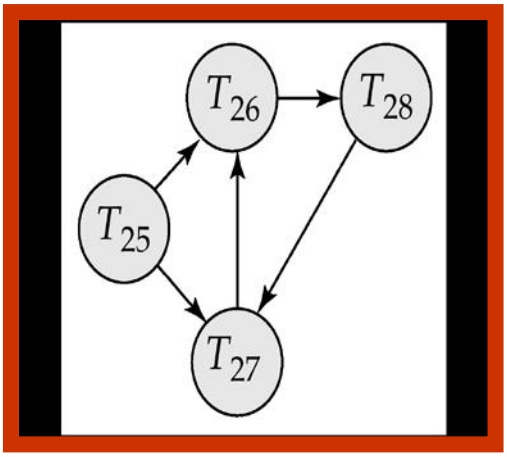
Timeout-Based Schemes :

- ★ a transaction waits for a lock only for a specified amount of time. After that, the wait times out and the transaction is rolled back.
- ★ thus deadlocks are not possible
- ★ simple to implement; but starvation is possible. Also difficult to determine good value of the timeout interval.

Deadlock Detection (Cont.)



Wait-for graph without a cycle



Wait-for graph with a cycle

Deadlock Detection

Deadlocks can be described as a *wait-for graph*, which consists of a pair $G = (V,E)$,

- V is a set of vertices (all the transactions in the system)
- E is a set of edges; each element is an ordered pair $T_i \rightarrow T_j$.

If $T_i \rightarrow T_j$ is in E, then there is a directed edge from T_i to T_j, implying that T_i is waiting for T_j to release a data item.

When T_i requests a data item currently being held by T_j, then the edge T_i T_j is inserted in the wait-for graph. This edge is removed only when T_j is no longer holding a data item needed by T_i.

The system is in a deadlock state if and only if the wait-for graph has a cycle. Must invoke a deadlock-detection algorithm periodically to look for cycles.

Recovery from Deadlock:

When a detection algorithm determines that a deadlock exists, the system must **recover** from the deadlock. The most common solution is to roll back one or more transactions to break the

deadlock. Three actions need to be taken:

1. Selection of a victim. Given a set of deadlocked transactions, we must determine which transaction (or transactions) to roll back to break the deadlock. We should roll back those transactions that will incur the minimum cost. Unfortunately, the term *minimum cost* is not a precise one. Many factors may determine the cost of a rollback, including

a. How long the transaction has computed, and how much longer the transaction will compute before it completes its designated task.

b. How many data items the transaction has used.

c. How many more data items the transaction needs for it to complete.

d. How many transactions will be involved in the rollback.

2. Rollback. Once we have decided that a particular transaction must be rolled back, we must determine how far this transaction should be rolled back. The simplest solution is a **total rollback**: Abort the transaction and then restart it. However, it is more effective to roll back the transaction only as far as necessary to break the deadlock. Such **partial rollback** requires the system to maintain additional information about the state of all the running transactions. Specially, the sequence of lock requests/grants and updates performed by the transaction needs to be recorded. The deadlock detection mechanism should decide which locks the selected transaction needs to release in order to break the deadlock. The selected transaction must be rolled back to the point where it obtained the .rst of these locks, undoing all actions it took after that point. The recovery mechanism must be capable of performing such partial rollbacks. Furthermore, the transactions must be capable of resuming execution after a partial rollback. See the bibliographical notes for relevant references.

3. Starvation. In a system where the selection of victims is based primarily on cost factors, it may happen that the same transaction is always picked as a victim. As a result,

this transaction never completes its designated task, thus there is **starvation**. We must ensure that transaction can be picked as a victim only a (small) .nite number of times. The most common solution is to include the number of rollbacks in the cost factor.

The Phantom Phenomenon

Consider transaction *T*29 that executes the following SQL query on the bank database:

select sum(*balance*) **from** *account* **where** *branch-name* = ’Perryridge’

Transaction *T*29 requires access to all tuples of the *account* relation pertaining to the Perryridge branch.

Let *T*30 be a transaction that executes the following SQL insertion:

insert into *account* **values** (A-201, ’Perryridge’, 900)

Let *S* be a schedule involving *T*29 and *T*30. We expect there to be potential for a conflict for the following reasons:

- If *T*29 uses the tuple newly inserted by *T*30 in computing **sum**(*balance*), then

*T*29 read a value written by *T*30. Thus, in a serial schedule equivalent to *S*, *T*30 must come before *T*29.

- If *T*29 does not use the tuple newly inserted by *T*30 in computing **sum**(*balance*),

then in a serial schedule equivalent to *S*, *T*29 must come before *T*30.

The second of these two cases is curious. *T*29 and *T*30 do not access any tuple in common, yet they con.ictwith each other! In effect, *T*29 and *T*30 con.ict on a phantom tuple. If concurrency control is performed at the tuple granularity, this con.ict would go undetected. This problem is called the **phantom phenomenon**.

------THANK YOU----