UNIT 9 DATA MODELS

Structure

9.0 OBJECTIVES

In effective database design, data modeling plays an important role. Data modeling is the process of structuring of data requirements.

After reading this Unit, you will be able to:

- understand the role of data modeling in conceptual database design;
- distinguish between various data models and know the characteristics of each;
- become conversant with relational database technology; and
- comprehend the steps involved in designing databases.

9.1 INTRODUCTION

This Unit presents data modeling concepts and the role of the Entity-Relationship (ER) model in conceptual database design. Classical data models, viz., hierarchical model, network model and relational model have been discussed. Concepts of object-oriented approach have been introduced with illustrations.

Considering that Relational Database Management Systems (RDBMS) are still widely in use, relational database technology has been dealt with in details. The concepts of dependencies and normalisation have been elucidated with examples.

A step-by-step procedure for designing databases has been given. The Unit provides core information in understanding database management systems.

9.2 DATA MODELING

Data modeling is a process by which the data requirements of an organization or an application area are represented. Conceptual modeling is an important phase in designing a successful database application. The traditional approach is of using the entity-relationship model for this purpose. Enhanced ER or EER model is used for modeling object-oriented databases.

9.2.1 Entity-Relationship Model

The Entity-relationship model, developed by Chen in 1976-77, serves as an excellent tool in the database design process. It provides graphic representation of entities, attributes and relationships. Requirements analysis of the designed database helps in collecting the necessary information in the form of entities and attributes to be included in the database. Based on enterprise rules, the relationships between the entities get identified and the nature of use of the database is determined. The E-R model describes the conceptual schema and is considered as a blueprint of the database under design. After finalisation, an E-R diagram (as the entity-relationship model is generally called) is mapped into one of the selected database models (discussed later in the text) and the system-dependent procedure of database creation is started.

An illustration of the E-R diagram has been given in Fig. 3.1. It depicts a database on marketing of drugs from medicinal and aromatic plants. The plants or their parts serve as crude drugs which are traded in the market. The standardising agency certifies the quality of drugs while the certifying agency approves the drugs for export. Before supply to the customer, the crude drugs are sometimes processed. In an ER diagram rectangles represent entities, ellipses show attributes, diamonds represent relationships, attributes with underscore show primary keys, attributes with double underscore represent foreign keys and 1, n, m show relationship types.

9.2.2 Types of Relationship

A relationship is an association between two or more entities. Entities correspond to record types in a database, which are sets. Thus a relationship represents a correspondence between the elements of n sets. The relationship over two sets is called a binary relationship; the relationship over three sets ternary and over n sets n-ary relationship.

Relationships can themselves be treated as entities and assigned attributes (see Fig. 3.1). Relationships can be grouped into the following types:

i) 1:1 (one-to-one)

ii) 1:n (one-to-many)

iii) n:1 (many-to-one)

iv) n:m (many-to-many)

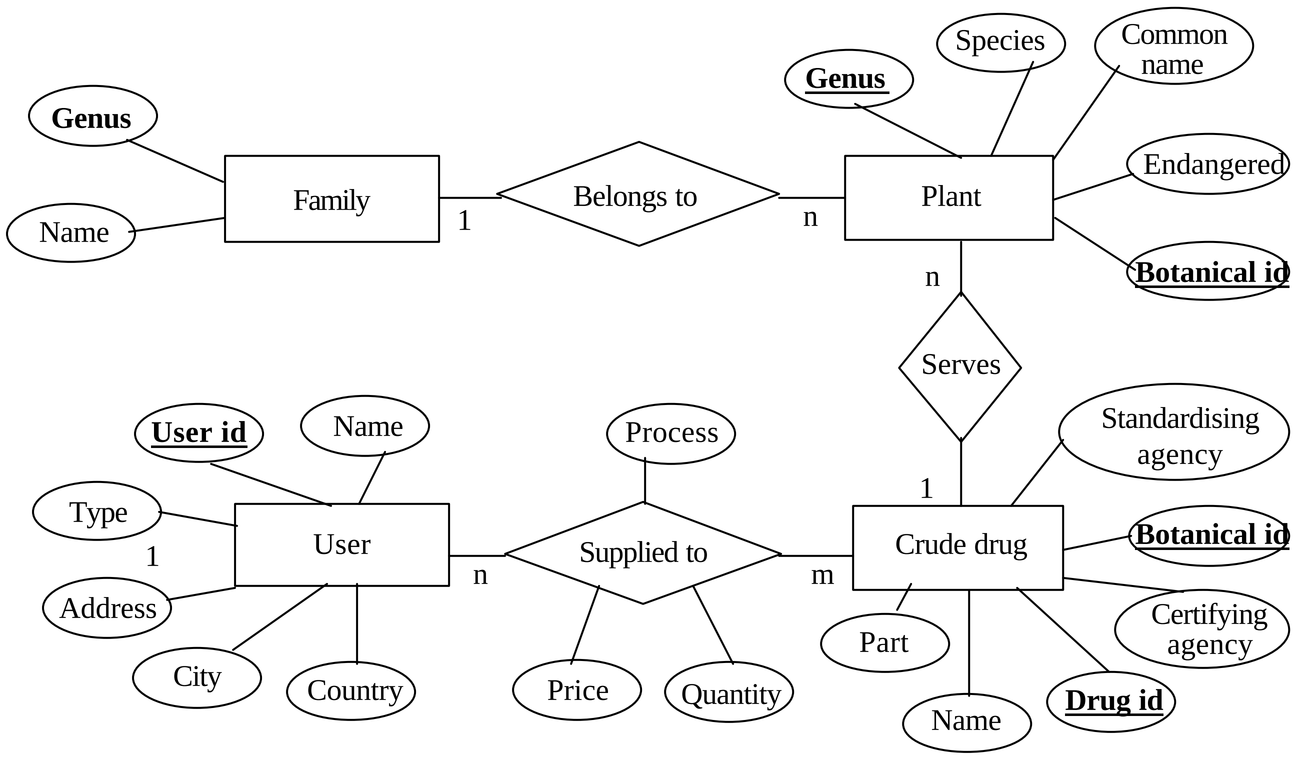


Fig. 3.1: Illustration of an E-R diagram

Let us illustrate these relationship types one by one. In a 1:1 relationship, one instance of an entity of a given type is associated with only one member of another type. Let there be a set of country names and a set of city names. Further let us assume that each city in the set is a capital. The relationship between these two sets which can be called "capital" is 1:1 because for each country name there is only one city name and, conversely, each city name corresponds to only one country name.

In a 1:n relationship, one instance of a given type of entity is related to many instances of another type. Let there be a set of departments and a set of faculty members employed in the departments. The department-faculty relationship which can be called 'employed' is of the 1:n type because each department employs several faculty members and each faculty member works only in one department.

The many-to-one (n:1) relationship has the same semantics as 1:n. In the above example, if we change the relationship to faculty-department (in place of department-faculty) we will have n:1 relationship type.

Lastly the n:m relationship is one in which many instances of an entity type are associated with many instances of another entity type. Consider a set of faculty members teaching a set of students. The faculty-student relationship ("teaching") is an example of the n:m type relationship because a faculty member can teach `m` students and a student can be taught by 'n' faculty members.

Self Check Exercise

1) What is the significance of an entity-relationship (ER) diagram ?

....

....

....

....

9.3 DATA MODELS

Data models are a methods by which data is structured to represent the real world and the manner in which the data is accessed. These models provide alternative ways of picturing the relationships and serve as frameworks for mapping the conceptual schema of a database. There are a number of data models in use today but the following three have been most widely implemented :

— Hierarchical

— Network

— Relational

Besides these three models which are sometimes referred to as classical models, post-relational research has resulted in a new data model called the object-oriented data model.

9.3.1 Hierarchical Model

The hierarchical model is the oldest of the three models. This model structures data so that each element is subordinate to another in a strict hierarchical manner.

The hierarchical model represents a 1:n relationship between record types (see Fig. 3.2). One record type (the 1 in 1:n relationship) is designated as the "parent" record type. In this parent-child relationship, a child record type can have only one parent record type but a parent record may have several child record types.

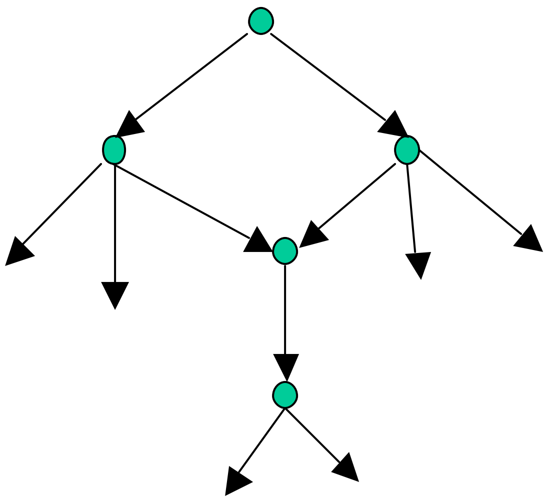


Fig. 3.2: A hierarchy (1:n relationship)

The hierarchical model is implemented by storing data in physical adjacency and using pointers to the dependent child records. The model suffers from undesirable data redundancy.

An example of the hierarchical model has been illustrated in Fig. 3.3. Here a hierarchy has been shown between two record types—one having fields like name, address, profession, account number, and the other giving account number and balance. Redundancy has been shown by an arrow.

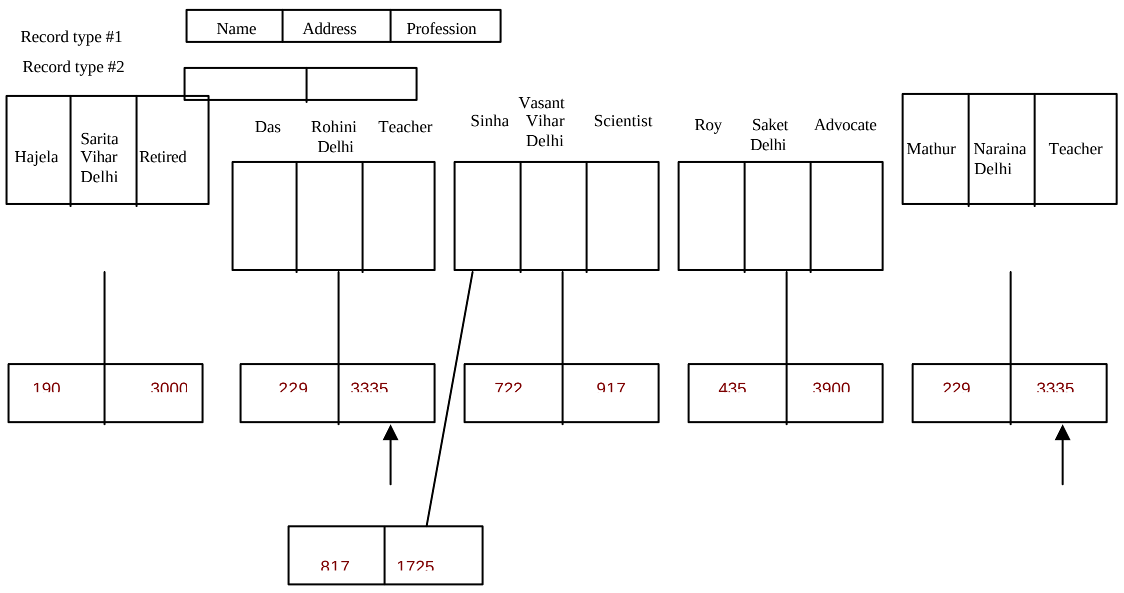


Fig. 3.3: An illustration of the hierarchical model

Examples of commercial implementation of the hierarchical model are IBM's IMS database management system and CDS/ISIS – a popular database management system for bibliographic applications.

9.3.2 Network Model

In the network model the restriction that a child can have only one parent record is removed. The network supports an n:m relationship (see Fig.3.4). A network can be a hierarchy (1:n being a special case of n:m) but a hierarchy cannot be a network.

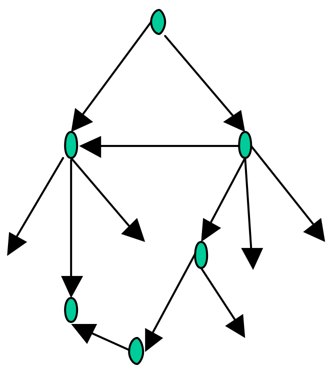


Fig. 3.4: A network (n:m relationship)

The network model is implemented with various pointer schemes. Since a network is an extension of hierarchy, the semantic properties of the network model are similar to those of the hierarchical model.The network model has been illustrated in Fig.3.5.

The example given makes use of the same record types as in the hierarchical model with the difference that the first record type has pointers to the A/c No. of the second record type. A pointer may itself point to a number of pointers (called arrays).

Cullinet's IDMS is an example of a commercial database management system based on the network model.

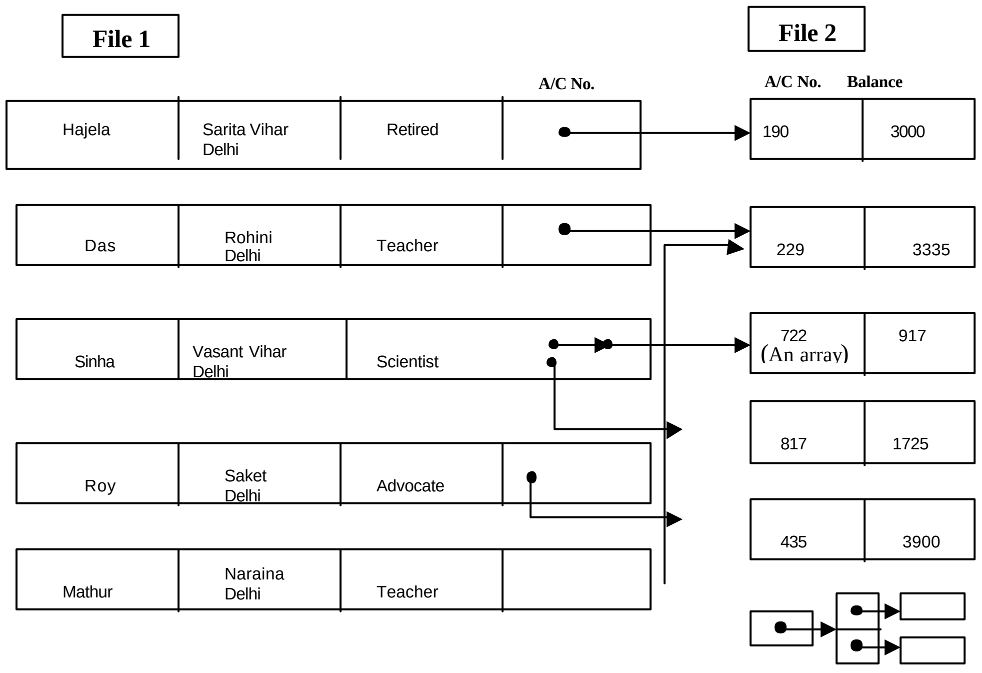


Fig. 3.5 : An illustration of the network model

9.3.3 Relational Model

The relational model maintains data in a tabular form which users find comfortable and familiar. The model is based on the well-developed mathematical theory of relations from which it derives its name. The name has nothing to do with the fact that the data stored in the relations is related (which usually it is).

The relational model supports the 1:1, 1:n, n:1 and n:m relationships. A significant aspect of the relational model is that the relationships between data items are not explicitly stated by pointers. Instead it is up to the DBMS to deduce the relationships from the existence of matching data in different tables. The absence of physical links provides a more flexible data manipulation environment.

An illustration of the relational model has been given in Fig. 3.6. Here the relationship between the two tables has been captured by repeating a column (A/c No.) in the first table.

ORACLE, INGRESS, and SYBASE are some of the well-known commercial database management systems based on the relational model.

Table name : Customer

Name	Address	Profession	A/C No.
Hajela	Sarita Vihar Delhi	Retired	190
Das	Rohini Delhi	Teacher	229
Sinha	Vasant Vihar Delhi	Scientist	817
Sinha	Vasant Vihar Delhi	Scientist	722
Roy	Saket Delhi	Advocate	435
Mathur	Naraina Delhi	Teacher	229

Table name : Account

A/C No.	Balance
190	3000
229	3335
817	1725
722	917
435	3900

Fig. 3.6: An illustration of the relational model

9.3.4 Object-oriented Model

The object-oriented data model facilitates handling of objects rather than records. In an object-oriented model an entity is represented as an instance (object) of a class that has a set of properties and operations (methods) applied to the objects. A class represents an abstract data type and is a shell from which one can generate as many copies (called instances) as one wants. In object-oriented approach, the behaviour of an object is a part of its definition. The behaviour is described by a set of methods. The set of methods offered by an object to the others defines the object interface. A class and hence an object may inherit properties and methods from related classes. Objects and classes are dynamic and can be created at any time.

Viewing the data as objects instead of as records provides more flexibility and removes the need to normalize data.

Fig.3.7 gives a comparison between the conventional database approach and the object-oriented database approach.

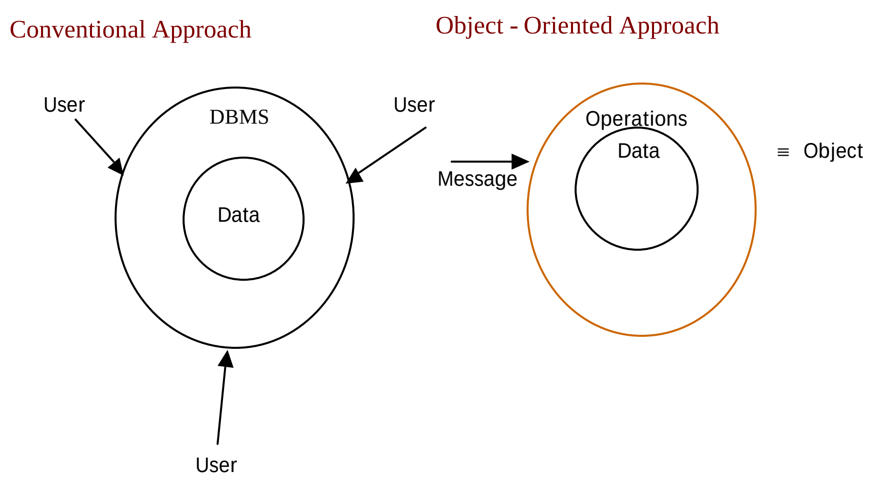


Fig. 3.7: Comparison between conventional and object-oriented database approaches

Some of the object building blocks have been defined below :

— Objects : An object is an entity, real or abstract, that has state, behaviour and identity. The state of an object is represented by its attributes and their values. The behaviour of an object is represented by its operations or methods.

— Messages : Objects communicate with each other through messages. A message determines what operation is to be performed by an object. A message specifies an operation name and a list of arguments.

— Classes : A class is a set of objects that share common attributes and behaviour. Each object is an instance of some class.

The object-oriented approach emphasises incremental software development. The underlying principle of this approach is:

- Grow software, don't build it
- Build components rather than a whole system
- Assemble a basic system and then enhance it.

Smalltalk, C++, Java and Object Pascal/Delphi are the object-oriented programming languages used in this approach.

Self Check Exercise

2) Why have RDBMS found wider application than other data models? Give a few examples of RDBMS.

....

....

....

....

9.4 RELATIONAL DATABASE MANAGEMENT SYSTEMS (RDBMS)

Relational database management systems have been evolving since the relational data model was first proposed in 1970 by Edgar F. Cod of IBM. They have become de facto international standard in database management. Despite great advances in the object-oriented database management systems, relational systems are likely to remain in vogue for quite some time.

Let us define some of the terms used in relational technology.

A relational database is a collection of data belonging to a system spread over a number of relations.

As pointed out earlier, relation is a mathematical concept. Mathematically a relation is a subset of the Cartesian product of a list of domains. A domain represents a set of values an attribute can assume (e.g., numeric, character etc.).

The degree of a relation determines the number of attributes in the relation. The number of tuples in a relation is called its cardinality. For example the customer table in Fig. 3.6 has four attributes and six tuples and hence the degree of this relation is four and cardinality six.

A relational database management system (RDBMS) is a collection of programmes that help to organise and manipulate data in a relational database.

The following terms are used interchangeably in RDBMS jargon :

— relation, table, database, file

— attribute, column, field

— tuple, row, record

— domain, range, type

9.4.1 Characteristics of a Relation

A relation exhibits the following characteristics :

i) A relation is always named.

ii) Each column of a relation has a unique name (although columns of different relations may have the same name).

iii) The order of the columns in a relation is of no consequence.

iv) There cannot be two identical tuples in a relation.

v) The order of the tuples in a relation is of no consequence.

vi) Each column of a relation has a domain specification.

vii) There may be more than one column in a relation having the same domain specifications.

viii) A column in a relation cannot have more than one domain specification.

ix) Each column contains values about the same attributes and each table cell value (intersection of a row and a column) must be single-valued.

The structure of a relation i.e., set the of attributes without any values assigned to them, is called the relation scheme. A tuple of a relation with values assigned to its attributes is called a relation instance.

A table is generally represented by its relation scheme which is denoted by the table name followed by the attribute names given in brackets. The relation schemes of tables shown in Fig. 3.6 are :

CUSTOMER (Name, Address, Profession, A/c No.)

ACCOUNT (A/c No., Balance).

The primary key attribute is underlined in the relation scheme.

9.4.2 Keys and their Functions

Keys play an important role in relational database management systems. They serve two basic purposes :

i) Identification of tuples

ii) Establishing relationship between tables

Keys (data items) can be made up of single attributes called single key attribute or multiple attributes called multikey attributes. Keys formed by multiple attributes are also called *composite or concatenated* keys.

A *superkey* is a set of attributes which taken collectively allows us to identify uniquely an entity in an entity set. If any key is a superkey, a superset of superkey is also a superkey. (If X1 is a subset of a set X, the X is called superset of X1).

The smallest superkey, which is also called *minimal key* is a key such that no proper subset of it is a superkey. One of the minimal keys is chosen as the *primary key*. The keys in the set of minimal keys are called *candidate* or *alternate* keys. It is up to the database designer to select one of the candidate keys as a primary key.

A primary key is an attribute or a combination of attributes that uniquely identifies a record while a *secondary key* does not identify a record uniquely. Secondary key identifies all the records corresponding to the key value.

A *foreign key* is an attribute or a combination of attributes that is used to link tables. In relational database management systems foreign keys are used as linking pins between tables. Foreign keys represent links to the primary keys.

Primary and foreign keys are critical in relational database management systems due to their contribution in defining integrity rules. A paramount guideline in relational systems is that a primary key or any attribute participating in a composite primary key of a relation cannot have null value. This rule is called entity integrity.

There is another integrity rule which pertains to foreign keys. According to this rule an attribute that is a foreign key in one table must be a primary key in another table. This rule is known as referential integrity.

9.4.3 Criteria for a DBMS to be Relational

Distinguishing a truly relational DBMS from relational-like systems assumes importance in view of the fact that many new DBMS packages are being labelled as “relational”. The minimum conditions for a system to be called relational are :

i) The data should be represented in the form of tables.

ii) Any pointer mechanism should be transparent to the DBMS users.

iii) The system should support relational algebra operators of SELECT, PROJECT and JOIN.

Any system which fulfills the above three criteria is called *minimally relational*. A system which satisfies only the first two conditions is not a relational system and is called *tabular DBMS*.

For a DBMS to be fully relational it should additionally support both entity and referential integrity rules and implement all relational algebra operations.

The founder of relational database theory, E.F. Codd, has outlined 12 rules that define a fully relational database system. These rules are based on the premise that a relational database management system should be able to manage databases entirely through its relational capabilities.

9.5 NORMALIZATION OF RELATIONS

Normalization of relations is an important aspect in database design which deals with the semantics of the data. It is a technique that structures data in such a manner that there are no anomalies in the database. Anomalies refer to undesirable side effects which can occur in relations resulting in poor database design. The process of normalization usually involves decomposing a relation into two or more relations with fewer attributes, i.e., taking a vertical subset of the parent relation. The criteria used to split relations determine the levels of normalization called *normal forms.*

It ought to be noted that normalization is primarily aimed at preventing or reducing data maintenance problems rather than improving the retrieval efficiency. Converting relations to normal forms and utilizing the join of the decomposed relations to retrieve data that could have been retrieved from one original table does not augur well for retrieval speed. Thus while normalizing relations we are sacrificing retrieval speed to improve the integrity, consistency and overall maintenance of data stored in the database.

As indicated earlier normalization of relations removes anomalies in the database. The anomalies can be categorised as :

— Insertion anomalies

— Deletion anomalies

— Update anomalies

An *insertion anomaly* occurs when we are unable to insert a tuple into a table. Such a situation can arise when the value of the primary key is not known. As per the entity integrity rule, the primary key cannot have null value. Therefore the value/s corresponding to primary key attribute/s of the tuple must be assigned before inserting the tuple. If these values are unknown, the tuple cannot be inserted into the table.

In case of a *deletion anomaly,* the deletion of a tuple causes problems in the database. This can happen when we delete a tuple which contains an important piece of information, and the tuple is the last one in the table containing the information. With the deletion of the tuple the important piece of information also gets removed from the database.

An *update anomaly* occurs when we have a lot of redundancy in our data. Due to redundancy, data updating becomes cumbersome. If we have to update one attribute value which is occurring a number of times, we have to search for every occurrence of that value and then change it.

The anomalies will be further elaborated when we discuss the normal forms.

Before we proceed with discussion on normalization it would be useful to understand the concept of dependencies.

9.5.1 Dependencies

A dependency refers to the relationship amongst attributes. These attributes may belong to the same relation or different relations. Dependencies can be of various types viz., functional dependencies, transitive dependencies, multivalued dependencies, join dependencies, etc. We shall briefly examine some of these dependencies.

Functional dependency (F.D) – Functional dependency represents semantic association between attributes. If a value of an attribute A determines the value of another attribute B, we say B is functionally dependent on A. This is denoted by

A $\rightarrow$ **B** and read as "A determines B" and A is called the determinant.

It should be noted that when a data item (or collection of data items) determines another data item, the second data item does not necessarily determine the first. An attribute is said to be fully functionally dependent on a combination of attributes if it is dependent on the combination of attributes and not functionally dependent on any proper subset of the combination. To illustrate functional dependency let us consider a relation with the following relation scheme:

BOOK (Book-id, Subject, Year of publication, Price). Book-id (Book identifier or identification number) is the primary key of the relation and hence determines subject, year of publication and price.

We can represent it as :

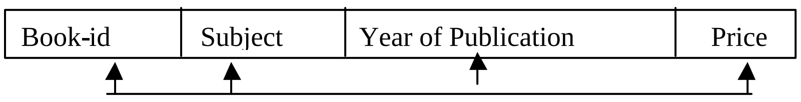
Book-id $\rightarrow$ Subject

Book-id $\rightarrow$ Year of publication

Book-id $\rightarrow$ Price

This can also be shown graphically in the form of a dependency diagram :

Book



Transitive dependency – Transitive dependency is a form of intermediate dependency. For example, if we have attributes or groups of attributes A, B and C such that A determines B and B determines C, i.e.,

A $\rightarrow$ B

B $\rightarrow$ C

Then we say a transitive dependency represented by A $\rightarrow$ B $\rightarrow$ C exists between A and C.

Let us consider a relation FACULTY.

FACULTY (Faculty-id, Name, Department, Office, Salary)

In this relation let us presume that each department has its own office building. Then, beside others, we have the following functional dependencies :

Faculty-id $\rightarrow$ Department (by virtue of Faculty-id being the primary key)

Department $\rightarrow$ Office (by virtue of the enterprise rule or constraint imposed by us on the relation)

Thus we have the following transitive dependency in the relation :

Faculty-id $\rightarrow$ Department $\rightarrow$ Office

Multivalued dependency — Multivalued dependency refers to m:n (many-to-many) relationships. We say multivalued dependency exists between two data items when one value of the first data item gives a collection of values of the second data item, i.e., it multidetermines the second data items.

Join dependency — If we decompose a relation into smaller relations and the join of the smaller relations does not give us tuples as in the parent relation, we say the relation has join dependency.

Let us consider a relation SAMPLE with the functional dependencies as shown :

SAMPLE

A	B	C
a1	b1	c1
a2	b2	c3
a3	b1	c2
a4	b2	c4

F.D : $A \rightarrow B$

$C \rightarrow B$

Let us now decompose this relation into two relations SPLIT 1 (A, B) and SPLIT 2(B, C). The functional dependencies will remain the same in the relation SAMPLE.

SPLIT 1

A	B
a1	b1
a2	b2
a3	b1
a4	b2

F.D. : $A \rightarrow B$

SPLIT 2

B	C
b1	c1
b2	c3
b1	c2
b2	c4

F.D.: $C \rightarrow B$

Now if we join the relations SPLIT 1 and SPLIT 2 over the common attribute B, we get the relation SAMPLE 1.

SAMPLE 1

A	B	C
a1	b1	c1
a1	b1	c2*
a2	b2	c3
a2	b2	c4*
a3	b1	c2
a3	b1	c1*
a4	b2	c3*
a4	b2	c4

We can see from the relation SAMPLE 1 that it has four additional (spurious) tuples (shown by asterisk) which were not present in the original relation SAMPLE. This type of join is called lossy join because the information content of the original table is lost. This has occurred since the join attribute was not the determinant in the original relation and hence it should not have been decomposed the way it was done. We say a join dependency exists in this case. If we decompose a relation and join the constituent relations over the determinant attribute, we get lossless join.

For example consider the relation SAMPLENEW and split it into SAMPLENEW1 and SAMPLENEW2 as given below :

SAMPLENEW

X	Y	Z	
x1	y1	z1	
x2	y2	z2	F.D: $X \rightarrow Y$
			$X \rightarrow Z$
x3	y2	z1	
x4	y1	z2	

SAMPLENEW1

X	Y	F.D : $X \rightarrow Y$
x1	y1	
x2	y2	
x3	y2	
x4	y1	

SAMPLENEW2

X	Z	F.D : $X \rightarrow Z$
x1	z1	
x2	z2	
x3	z1	
x4	z2	

The join of SAMPLENEW1 and SAMPLENEW2 gives the original table SAMPLENEW without any spurious rows because the attribute X over which the table is decomposed is the determinant attribute.

9.5.2 Normal Forms

The technique of normalization is based on the analysis of functional dependencies between attributes. Taking into account the total scenario, enterprise rules and semantic constraints, all the functional dependencies in the relations are captured and the relations transformed into proper normal forms. A normal form represents the level of normalization of a relation. Normalization proceeds by converting a relation from a lower normal form to a higher normal form.

The norm forms are :

First normal form (1NF), second normal form (2NF), third normal form (3NF), Boyce-Codd normal form (BCNF), fourth normal form (4NF), fifth normal form (5NF) and the highest normal form which is called domain/key normal form (DK/ NF).

E.F. Codd had outlined the first three normal forms which we shall discuss in detail. Most of the commercially available DBMS are normalized up to 3NF or BCNF.

Normal forms build on each other, i.e., if a relation is in 3NF, it is also in 1NF and 2NF.

The first normal form (1NF). A relation is in the first normal form if it can be represented as a flat file, i.e., the relation contains single values at the intersection of each row and column.

In other words each attribute value in a tuple is atomic, i.e., non-decomposable.

Let us consider a relation PERSON which is not in 1NF.

PERSON (<u>Name</u>, Age, Gender, $C_{Name, Age}$)

In this relation $C_{Name, Age}$ pertains to the attribute dependent child with multiple values (name and age). To convert this relation into 1NF, it should be decomposed into two relations as follows :

PERSON (<u>Name</u>, Age, Gender)

DEPENDENT (Name, C_{Name}, C_{Age})

The second normal form (2NF). A relation is in the second normal form if it is in 1NF and every non-key attribute is fully functionally dependent on the primary key. The second normal form pertains only to relations with composite primary key. In case a relation is in 1NF and has a single-attribute primary key, it is automatically in the 2NF.

To explain the second normal form let us take the example of the following relation.

COURSE (<u>Course-id</u>, Course-name, Class-number, <u>Student-id</u>, <u>Faculty-id</u>).

In this relation Course-id, Student-id and Faculty-id form a composite primary key.

The following functional dependencies can be noticed in this relation.

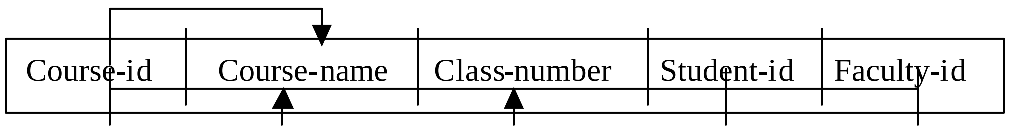
Course-id, Student-id, Faculty-id $\rightarrow$ Course-name

Course-id, Student-id, Faculty-id $\rightarrow$ Class-number

On examining the relation semantically we observe that Course-id uniquely determines Course-name i.e.,

Course-id $\rightarrow$ Course-name

This means that Course-name is not fully functionally dependent on the primary key. Therefore the relation is not in the 2NF. The dependency diagram of the relation can be represented as follows :



To convert this relation into the 2NF, we decompose it into two relations as follows:

COURSE (Course-id, Class-number, Student-id, Faculty-id)

COURSE-TITLE (Course-id, Course-name)

The decomposition is done by extracting the attribute that caused the problem from the relation and creating a new relation with this attribute.

Let us examine these relations (normalized and unnormalized) on the basis of anomalies described earlier.

COURSE (unnormalized)

Course-id	Course-name	Class-number	Student-id	Faculty-id
C701	Computer science	7	S009	B03
C701	Computer science	8	S008	G04
C702	Library automation	7	S009	A01
C702	Library automation	8	S006	G04
E500	Microeconomics	7	S009	P02
E501	Macroeconomics	7	S001	P02
M200	Management studies	7	S001	V01

The normalized relations obtained from the above relation are :

COURSE

Course-id	Class-number	Student-id	Faculty-id
C701	7	S009	B03
C701	8	S008	G04
C702	7	S009	A01
E500	8	S006	G04
E501	7	S007	P02
M200	7	S001	V01

COURSE-TITLE

Course-id	Course-name
C701	Computer science
C702	Library automation
E500	Microeconomics
E501	Macroeconomics
M200	Management studies

Insertion: In the case of unnormalized relation, suppose a new course named ‘Computer networks’ has to be introduced. We cannot enter the Course-name in the relation, since we have no means to ascertain Student-id and Faculty-id at the time of introduction of the course and these attributes which along with Course-id form the primary key cannot be assigned null values. However, in the normalized relation COURSE-TITLE this problem does not arise.

Deletion: Suppose in the unnormalized relation the student with Student-id S001 leaves the course. This student being the only student in the particular course, deletion of the tuple would result in loss of information and we will have no way to know that the course 'Management Studies' exists. In normalized relation the deletion of this tuple does not result in information loss because the information about the course name exists in the second relation.

Updating: Let us assume that the name of the course 'Library automation' changes to 'Library management'. In the unnormalized relation we will have to change the information in two tuples while in the normalized relation only one tuple would require to be updated. This may not appear to be a significant gain but if the database size is large the problem can have serious repercussions.

Third normal form (3NF). A relation is in the third normal form if it is in second normal form and contains no transitive dependencies. This normal form is considered to be the most important normal form.

To illustrate 3NF, let us consider the following relation :

FACULTY (<u>Faculty-id</u>, Faculty-name, Department, Gender, Salary, Office)

Let us also assume that each department has its office in one building.

This relation has the following functional dependencies by virtue of Faculty-id being the primary key.

Faculty-id $\rightarrow$ Faculty-name

Faculty-id $\rightarrow$ Department

Faculty-id $\rightarrow$ Gender

Faculty-id $\rightarrow$ Office

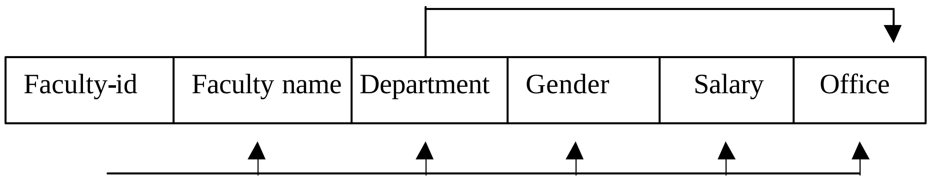
We also know from the semantics of this relation that

Department $\rightarrow$ Office

Therefore we have a transitive dependency as shown below :

Faculty-id $\rightarrow$ Department $\rightarrow$ Office

Thus dependency diagram of the relation can be represented as :



To convert this relation into the 3NF we shall have to decompose it into two smaller relations. The new relation OFFICE-NAME has the attribute office which caused transitive dependency and its determinant. The decomposed relations are :

FACULTY (<u>Faculty-id</u>, Faculty-name, Department, Gender, Salary)

OFFICE-NAME (<u>Department</u>, Office)

With some tuple values in these relations let us examine them for anomalies which are used to test relations.

FACULTY (unnormalized)

Faculty-id	Faculty-name	Department	Gender	Salary	Office
B03	Bindra	Computer Science	M	4000	Birla Block
G04	Ganguly	Computer Science	M	4500	Birla Block
A01	Arora	Computer Science	F	3450	Birla Block
P02	Pandey	Economics	F	3500	Kanishka Block
V01	Vohra	Mathematics	M	4500	Ashoka Block
B01	Bansal	Physics	M	3000	Ashoka Block

The normalized relations are :

FACULTY

Faculty-id	Faculty-name	Department	Gender	Salary
B03	Bindra	Computer Science	M	4000
G04	Ganguly	Computer Science	M	4500
A01	Arora	Computer Science	F	3450
P02	Pandey	Economics	F	3500
V01	Vohra	Mathematics	M	4500
B01	Bansal	Physics	M	3000

OFFICE-NAME

Department	Office
Computer Science	Birla Block
Economics	Kanishka Block
Mathematics	Ashoka Block
Physics	Ashoka Block

Insertion : Let us assume that a new department whose faculty has not yet been finalized is created. We cannot enter this information in the unnormalized relation because we do not know value of the primary key (Faculty-id). To assign values to Faculty-id corresponding to new department we must know the values of other attributes (Faculty-name, Gender, Salary, Office). This problems does not exist in the normalized relation.

Deletion : Suppose a faculty member Vohra of the mathematics department leaves the faculty. If we delete tuple corresponding to this Faculty-name in the unnormalized relation, the information that the mathematics department exists is also lost. However, this does not happen in the normalized relations.

Update : If the office of the Computer Science department shifts from Birla Block to Nehru Block, 3 tuples need to be updated in the unnormalized relation while only one tuple would be modified in normalized relation.

Boyce-Codd normal form (BCNF). Originally normal forms stopped at the 3NF. However, research into dependencies led to higher normal forms. BCNF is an extension of the third normal form. It states that if a relation is in 3NF and all determinants are candidate keys, then it is in Boyce-Codd normal form.

The fourth normal form (4NF) refers to multivalued dependencies. A relation is said to be in the fourth normal form if it has only one multivalued dependency.

The fifth normal form (5NF) is also called project join normal form (PJNF). If a relation is in 5NF we should be able to join the projections (decompositions) of the relation and reconstruct the original relation without any information loss.

The domain key normal form (DK/NF) is considered the highest normal form. A relation is in DK/NF if every constrain can be inferred by simply knowing the set of attribute names and their underlying domain along with their set of keys. Thus in DK/NF only two types of constraints are allowed – domain constraints and key constraints. If these constraints are fully enforced other constraints (dependencies) are removed and no anomalies can occur in the relation.

The process of normalization from lower to higher normal forms has been illustrated in Fig. 3.8.

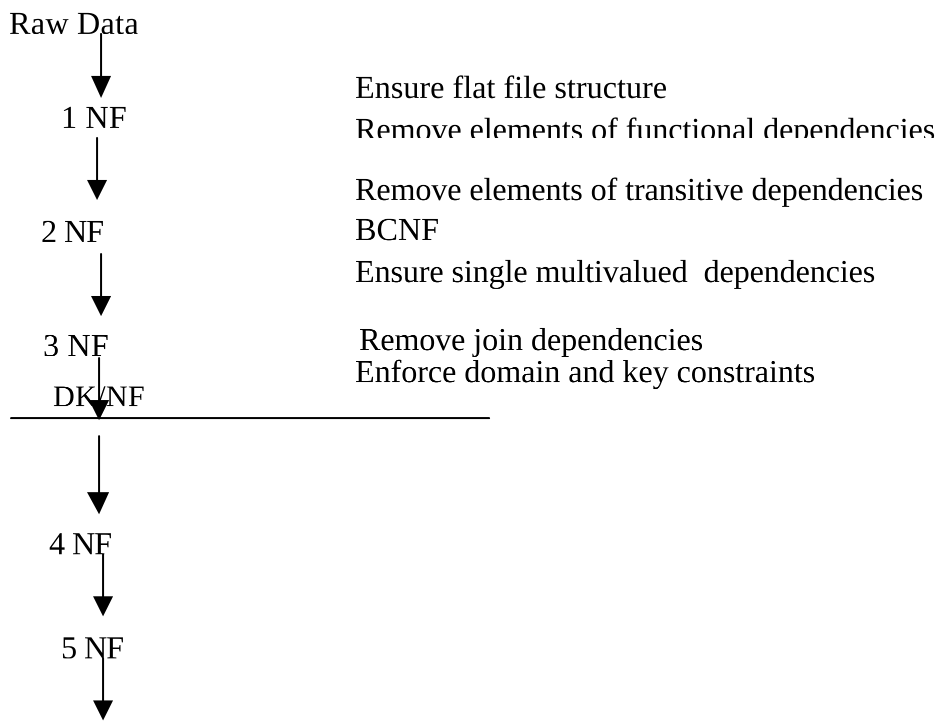

Raw Data
1 NF
2 NF
3 NF
DK/NF
4 NF
5 NF
Ensure flat file structure
Remove elements of functional dependencies
Remove elements of transitive dependencies
BCNF
Ensure single multivalued dependencies
Remove join dependencies
Enforce domain and key constraints

Fig. 3.8: Normalization process

Self Check Exercise

3) What are the advantages and disadvantages of normalization of relations?

....

....

....

....

....

9.6 DESIGNING DATABASES

Designing a database is a highly complex operation. Though it is relatively easy to identify a poorly designed database, there does not exist a unified approach which leads to the best design.

The database design should be flexible enough to meet the requirements of the maximum number of users to the fullest. Besides, the design should also anticipate, to a certain extent, future requirements and make provisions for them. This calls for some intuitiveness on the part of database designer.

The process of designing a database is an interactive one which means that the initial database structure changes with usage. However, with time the design tends to get stabilized. Usually a person designated as database administrator (DBA) controls the design and administration of a database.

A broad step-by-step procedure for designing a database has been summarized below:

1) First of all data to be represented in the database is determined. For this, the information needs of the users are studied in detail. Based on the information requirements analysis, entities of interest are identified and their attributes examined.

2) An E-R model of the database representing a conceptual schema is drawn. This is the most important stage in the database design process. The E-R diagram which depicts entitites and their relationships should be as comprehensive as possible.

3) The E-R model is mapped into a selected database structure (hierarchical, network, relational or any other model). In case a relational model is chosen, tables corresponding to entities and their relations are finalized. The process of normalization is invoked to check the tables and reshape them if necessary.

4) An empty database is created using DBMS commands (create TABLE, INDEX, etc.). A data dictionary which defines data item names and their internal storage format is also created by DBMS. The database design process up to step 3 is system-independent. The process becomes system-dependent after that.

5) The database is populated. This involves inserting data into the empty database. If data to be inserted is available in machine-readable form, the data loading utility of DBMS can be utilised.

6) The performance of the database is closely monitored to ascertain whether any tuning is required. Flexibility and speed of access are critically evaluated. The database is also examined for data maintenance problems.

7) The feedback of the users on the database functionality is analysed and changes in the structure made to optimise the usage.

9.7 SUMMARY

The E-R model plays an important role in conceptual database design. The evolutionary path of data from hierarchical model to relational and then to object-oriented has brought about tremendous changes in database design techniques. Relational database management systems are by far the most popular. Normalisation of relations is an important aspect of database design aimed to remove anomalies in a database. It improves integrity and consistency of data, though it slows retrieval speed.

Designing databases is a highly complex process. There are a number of basic steps which a database designer follows while designing databases. Usually a database stabilizes in design over a period of time with feedback from users.

9.8 ANSWERS TO SELF CHECK EXERCISES

1) The E-R diagram is a tool that models the relationships among the entities in a database. It maps the conceptual schema and serves as a blueprint for designing databases.

2) Relational database management systems (RDBMS) have become the de facto international standard in database management. Despite great advances in the object-oriented systems and other systems, relational systems retain their wide acceptance.

   RDBMS have advantages over other data models in the fact that the relational model is based on the well-developed mathematical theory of relations from which it derives its name. Application of mathematics imparts great strength to the relational model. The data in relational systems is represented in the form of tables which users find easier to handle. Examples of RDBMS are : ORACLE, SYBASE and INGRESS.

3) The advantages of normalisation of relations are that it enforces data integrity and removes anomalies in a database. It also minimizes data redundancy and in general promotes accuracy and consistency of data in the database.

   Since the process of normalization involves decomposing a table into two or more tables, which are joined while retrieving data, the retrieval speed gets adversely affected.

9.9 KEYWORDS

Dependency	**:**	A dependency refers to the relationship amongst attributes elonging to the relation or different relations.
E-R Diagram	**:**	Entity-Relationship Diagram. A diagram that shows associations (relationships) between entities.
Foreign Key	**:**	A column in one table that is the primary key in a second table. It does not need to be a key in the first table.
Normalization	**:**	The process of creating a well-behaved set of tables to efficiently store data, minimize redundancy, and ensure data integrity.

Primary Key	**:**	A column or a set of columns that identify a particular row in a table.
Relation	**:**	A relation is a table.
Relationship	**:**	An association between two or more entities.

9.10 REFERENCES AND FURTHER READING

Claude Delobel, Michel Adiba (1985). Relational Database Systems. Amsterdam : Elsevier Science Publishers B.V.

Codd, E.F. (1990). The Relational Model for Database Management : Version 2, Addison, New York : Wesley Publishing Company. Inc.

Date, C.J. (1989). Introduction to Database Systems. New Delhi : Narosa Publishing House.

James, F. Courtney, David, B. Paradice. (1988). Database Systems for Management. Toronto : Times Mirror/Mosby College Publishing.

James Martin (1988). Principles of Database Management. New Delhi: Prentice-Hall of India Private Limited.

Jeffrey, D. Ullman (1991). Principles of Database Systems. New Delhi : Galgotia Publications (P) Ltd.

Ramaz Elmasri, Shaukan B. Navathe (2000). Fundamentals of Database Systems: Pearson Education Asia.