COMPUTER SCIENCE

B. Sc. II (CBCS)
Semester-IV
2023-2024

2CS2 : RDBMS and Core Java

Unit-I : Fundamental of DBMS

PROF. V. V. AGARKAR

Assistant Professor & Head

Department of Computer Science

Shri. D. M. Burungale Science & Arts College, Shegaon, Dist. Buldana

UNIT - I

***Syllabus*: Fundamental of DBMS:** Traditional Vs DBMS File approach, DBMS Architecture, Data Models, Relational Model, Relations, Domain and Attributes, Keys, E-R diagram, reducing ER diagram to table, Functional Dependency, Normalization: 1NF, 2NF, 3NF, 4NF, BCNF.

Traditional vs DBMS File approach

The traditional file approach and the DataBase Management System (DBMS) approach are two different methods for organizing and managing data in computer systems. Following is a comparison between this two:

1. **Data Organization:**
   - **Traditional File Approach**: In traditional file processing systems, data is typically organized into separate files, where each file contains records.
   - **DBMS Approach**: In a DBMS, data is organized in a structured format using tables. Tables consist of rows and columns, where each row represents a record.
2. **Data Integrity:**
   - **Traditional File Approach**: Ensuring data integrity can be challenging in the traditional file approach, as there may be no built-in mechanisms for this.
   - **DBMS Approach**: DBMS provides mechanisms such as constraints, triggers, and transactions to enforce data integrity.
3. **Data Access:**
   - **Traditional File Approach**: In file systems, data access is often procedural, i.e. developers must write programs to read and manipulate data from files.
   - **DBMS Approach**: DBMS provides a query language (SQL) for accessing and manipulating data. Users can write queries to retrieve, insert, update, and delete data without needing to know the underlying data storage details.
4. **Data Redundancy:**
   - **Traditional File Approach**: Data redundancy is common in the traditional file approach because the same data may be stored in multiple files or within the same file. This can lead to data inconsistency.
   - **DBMS Approach**: DBMS aims to minimize data redundancy through normalization techniques. This helps maintain data integrity and consistency.
5. **Security:**
   - **Traditional File Approach**: In file systems, security mechanisms need to be implemented manually, which can be error-prone and complex.
   - **DBMS Approach**: DBMS provides security features such as authentication, authorization, and encryption to protect data from unauthorized access.

Database

A database is a well-organized collection of stored operational data used by the application system of some particular enterprise. An 'enterprise' may be any self contained commercial, scientific, technical, or other organization. Some examples are: - Manufacturing Company, Bank, Hospital, University, etc.

Data Base Management System

A **database management system** (DBMS) is the software that handles all access to the database. It is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the **database**. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

Advantages / Characteristic / Objectives of DBMS

1. **Mass Storage**

   DBMS can store a lot of data in it. So for all the big firms, it can store thousands of records in it and one can fetch all that data whenever it is needed.

2. **Removes Duplicity (Redundancy)**

   DBMS guarantee it that there will be no data duplicity among all the records. While storing new records, DBMS makes sure that same data was not inserted before.

3. **Multiple Users Access (Shareability)**

   DBMS is an ability to share data resources. Two or more users can access database simultaneously. DBMS makes it sure that multiple users can work concurrently.

4. **Data Protection**

   DBMS gives a master level security to their data. No one can alter or modify the information without the privilege of using that data.

5. **Data Backup and recovery**

   DBMS has the ability to backup and recover all the data in database, so that on database failure it can be recovered.

6. **Integrity**

   Integrity means your data is authentic and consistent. DBMS has various validity checks that make your data completely accurate and consistence.

7. **Platform Independent**

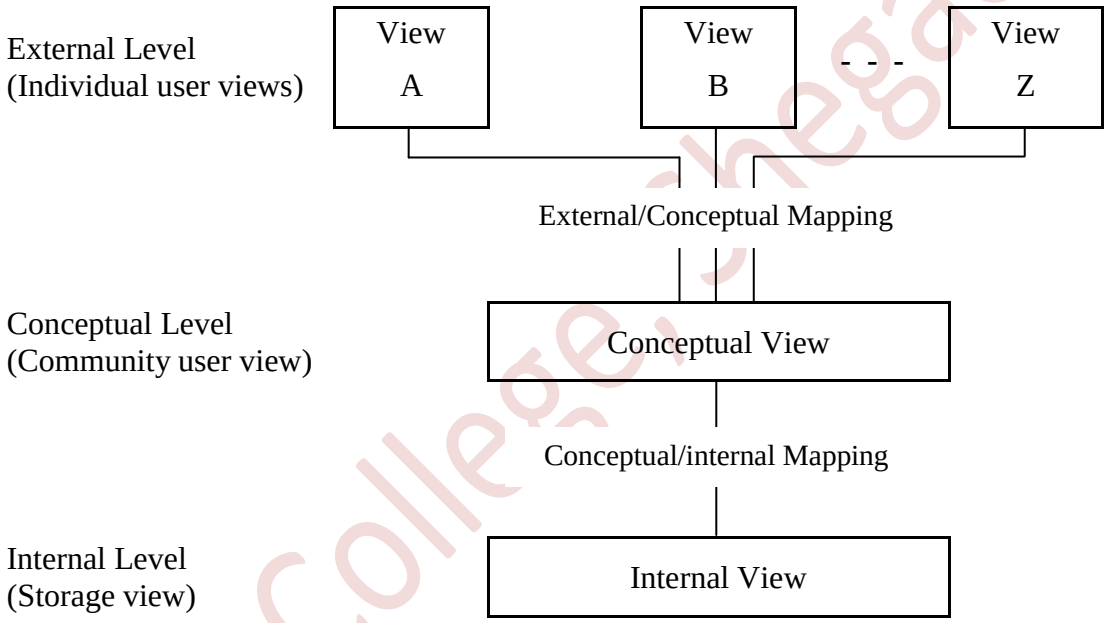
   One can run DBMS at any platform. No particular platform is required to work on database management system.

Architecture of DBMS (Three-tier architecture)

The objective of the architecture is to separate the users' view. This logical architecture describes how data in the database is viewed by users. It is not concerned with how the data is handled and processed by the DBMS, but only with how it looks.

The majority of DBMS available today are based on the ANSI/SPARC generalized architecture. Hence this is also called as the *ANSI/SPARC model*. It divides the system into three levels of abstraction: external level, conceptual level & internal level. Hence it is also called **three-level** or **three-tier** architecture.

The view at each of these levels is described by a ***schema***. A ***schema*** describes the logical records and relationships existing in the view. Following figure (**Fig.1**) shows the three levels of the database system architecture:



[**Fig. 1:** Three levels of database architecture]

1. External level (view)

This is a user's view of the database. It allows the user to see only the data of his interest and hides the data which the user is not authorised to access. This is the individual user level. There can be many external views of the database, so that the same data can be seen by different users in different ways, at the same time. The user at this level can be either an application programmer or an end user.

Each external view is defined by means of ***external schema***; hence there are many external schemas. Each external schema consists of definitions of logical record and relationships in the external view.

2. Conceptual level (view)

The conceptual level presents a logical view of the entire database as a whole. It allows the user to view all the data in the database together at one place. This level does not provide any storage or access details. DBA works at this level.

Prof. V. V. AGARKAR
D. M. Burungale Science & Arts College, Shegaon

The conceptual view is defined by means of the ***conceptual schema***, and there is only one conceptual schema per database.

3. Internal level (view)

The internal level is the one closest to the physical storage. It describes how the data is actually stored in the database and on the computer hardware. At this level the record types and methods of storage are defined.

The internal view is described by means of ***internal schema***. It not only defines the various types of stored records, but also specifies what indexes exist, how stored fields are represented what physical sequence the stored records are in, & so on.

4. External/conceptual mapping

The External/conceptual mapping is a mapping between the external view and the conceptual view. An external/conceptual mapping defines the correspondence between a particular external view and conceptual view. This mapping determines how the conceptual record is viewed by the user.

5. Conceptual/internal mapping

The conceptual/ internal mapping defines the correspondence between the conceptual view & the stored database. It specifies how conceptual records & fields are represented at the internal level. If the structure of the stored database is changed then the conceptual/ internal mapping must be changed accordingly, so that the conceptual schema can remain invariant.

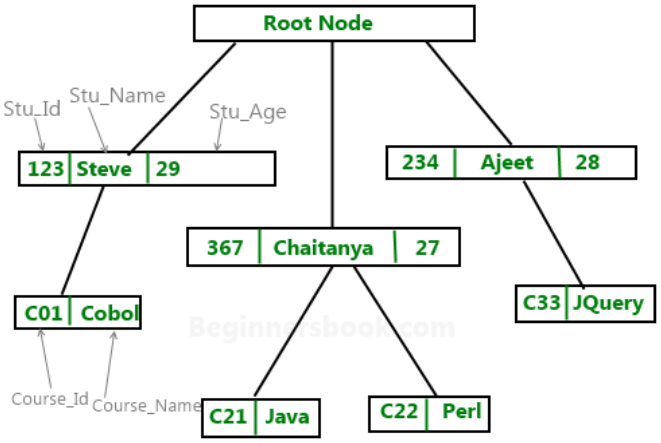
Data Model

In DBMS, a data model defines the way data is organized, documented, and defined within a database. A database model shows the logical structure of a database, including the relationships and constraints that determine how data can be stored and accessed. The data model also describes the elements used to standardize the system, such as associations, entities and requirements. There are several types of data models used in DBMS, Following are some types of data model:

1) Hierarchical data model
2) Network data model
3) Object-Oriented data model
4) Relational data model
5) Entity-Relationship (E-R) data model (*see page No. 9*)

1. Hierarchical data model

The hierarchical model is one of the oldest model which was developed by IBM, in the 1950s. In this model, the data is organized into a tree-like structure where each record consists of one parent record and many children. Sibling records are sorted in a particular order. That order is used as the physical order for storing the database. This model represent ***one-to-many*** relationship. This model is good for describing many real-world relationships. This model was primarily used by IBM's Information Management Systems in the 60s and 70s, but they are rarely seen today due to certain operational inefficiencies.

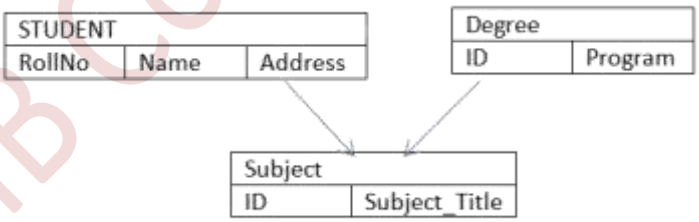


[**Fig. 2**: Hierarchical data model]

The main drawback of this model is that, it can have only one-to-many relationship between nodes.

2. Network data model

The network data model is a database model designed to represent complex relationships between data entities. It was developed in the 1960's as an enhanced form of the hierarchical data model. In the network model, data is organized as a collection of records and relationships are established through pointers or links between these records. Data organization in the network model is just like a graph rather than a tree. Unlike the hierarchical model, where each child node could have only one parent, in the network model, a child node can have multiple parent nodes, allowing ***many-to-many*** relationships. The network model was the most widely used model before the relational database model was introduced. Following figure is an example of a network model:



[**Fig. 3**: Network data model]

In this figure, the subject is the child class and student and degree are the parent classes. So, the subject has two parent classes.

3. Object-Oriented data model

The object-oriented data model is a way of structuring and representing data in which real-world entities are modeled as objects that have both data attributes (properties) and behaviors (methods). It is based on the principles of object-oriented programming (OOP), which include encapsulation, inheritance, and polymorphism. This model is widely used in software development for building complex, modular, and

maintainable systems. It promotes code reuse, modularity, and flexibility, making it well-suited for modeling and simulating real-world entities and systems.

4. Relational data model

The relational data model was introduced by Edgar F. Codd in the early 1970s and has become the most widely used data model for databases. This model is based on the concept of a ***relation***, which is essentially a ***table*** with rows and columns. A row is called as ***tupple*** and column is called as ***attribute***. The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Each table contains records of a particular type, each record type defines a fixed number of fields (attributes). The columns of the table correspond to the attributes of the record type. The relational model is an example of a record-based model.

Advantages of the relational data model include its simplicity, flexibility, and ability to handle complex relationships between data entities. It also provides a standardized way to query and manipulate data using Structured Query Language (SQL).

Example:

Student

StudentID	SName	DOA	DeptID
101	Smith	02-07-2024	001
102	John	04-07-2024	006
103	Blake	05-07-2024	002
104	David	05-07-2024	005
105	Mary	08-07-2024	003

Department

DeptID	DName
001	Chemistry
002	Computer
003	Electronics
004	Environment
005	Mathematics
006	Microbiology

In this example, the Student table stores information about students, while the Department table stores information about students admitted to the department. The DeptID column in the Student table is a foreign key that references the DeptID column in the Department table, establishing a relationship between the two tables.

Advantages

1) The relational query language gives a set of commands that allows the user to perform any number of operations.
2) No necessity of learning inner structure of the database.
3) Data independence allows easy structure modifications.
4) Set processing allows one command to modify large group of data.

Disadvantages

1) Query languages require more processing time and money.
2) Data must be related to act upon.
3) Set processing is not suited to conventional programming language.

Relation

Given a collection of sets $D_1, D_2, \ldots, D_n$, not necessarily distinct, R is a relation on those n sets, if it is a set of ordered n-tuples $<d_1, d_2, \ldots, d_n>$ such that d_1 belongs to D_1, d_2 belongs to D_2, . . ., d_n belongs to D_n. Sets $D_1, D_2, \ldots, D_n$ are the domains of R. The value n is the degree of R.

Consider a relation PART of degree 5 as shown below:

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London

[Fig.4: The relation PART **]**

Each row of the table represents one ***tuple*** & each column represents one ***attribute*** of the relation.

The number of attributes in the relation is called the ***degree*** of the relation and the number of tuples in the relation is called ***cardinality*** of the relation. The degree of PART relation is 5 and cardinality of PART relation is 4. Relations of degree one are said to be unary, relations of degree two are binary, … and relations of degree ***n*** are ***n-ary***.

Attribute

In relational database systems, ***attributes*** corresponds to fields (i.e. columns) of relation. An attribute may only be allowed to take a value from a set of permissible values. This set of allowable values for the attribute is called the ***domain*** of that attribute. Every attribute is defined on one underlying domain and values for any attributes are taken only from its corresponding domain.

Example :- In relation `PART` as shown in Fig.1, `P#`, `PNAME`, `COLOR`, `WEIGHT` and `CITY` are the attributes. Total five attributes are in relation `PART`.

Domain

The ***domain*** is a set of homogeneous members. The members from domain can be assigned to the attributes. Therefore for every attribute in a relation there is one underlying domain.

The domain is said to be *simple* or *atomic* if all its elements are non-decomposable (i.e. atomic). The domain is said to be *composite* or *non-atomic* if all its elements are non-atomic.

Example:- In relation `PART` as shown in Fig.1 is defined with five attributes namely, `P#`, `PNAME`, `COLOR`, `WEIGHT` and `CITY` so there are five domains. The five domains are sets of values representing, respectively, part numbers, part names, part colors, part weights, and locations in which parts are stored. . Each specified attribute is being drawn from a corresponding domain.

Keys

Key is an attribute or group of attributes, which is used to identify a row in a relation. Following are some significance of keys:

1. Keys ensure that each record (i.e. tuple) in the table is properly identified.
2. Keys help establish & enforce integrity rules.
3. Keys help establish relationship between tables (i.e. relations).
4. Keys enable faster searches in a table.

Following are different types of keys:

1. Primary Key

A primary key is the attribute which is unique within the relation. Primary key is used to uniquely identify any tuple in the relation. Following are the important elements of the primary key:

- It must have unique values and it cannot be a null field.
- Its value cannot be modified except in very rare cases.

For example, attribute S# of the S relation contains a distinct S# value, and can be used to distinguish that tuple from all others in the relation. Thus S# is said to be *primary key* for S.

2. Super Key or Combination key

Not every relation will have a single-attribute primary key. However, every relation will have some combination of attributes that, when taken together, have the unique identification property. Such keys are known as combination keys or super keys. In the relation SP, the combination (S#, P#) has this property.

3. Candidate Key

Occasionally in a relation, there is more than one attribute possessing the unique identification property, such attributes are called candidate keys and hence a relation have more than one candidate key.

For example, in S relation, each supplier has a unique S# and SNAME. Both S# and SNAME are the candidate keys of relation S. In such case one may arbitrarily choose one of the candidates, say S#, as the primary key for the relation.

4. Alternate key

A candidate key that is not a primary key is called alternate key. An alternate key is useful in providing a second means of identification for each tuple in the relation.

For example, in S relation, both S# and SNAME are the candidate keys. S# chooses as the primary key for the relation, and then SNAME is an alternate key.

5. Foreign Key

A foreign key is a copy of a primary key in another relation. The key connects to another relation when a relationship is being established.

For example, S# and P# keys of SP relation have this property.

Prof. V. V. AGARKAR
D. M. Burungale Science & Arts College, Shegaon

Entity-Relationship (E-R) Model

The Entity-Relationship (E-R) data model allows to describe the data involved in a real-world in terms of objects and their relationships. The E-R data model is based on the real world as a collection of basic objects called ***entities*** and the ***relationships*** among these objects. The E-R model expresses graphically by an E-R diagram.

- ***Entity / Entities***

An **entity** is a "thing" or "object" that exists and is distinguishable from other objects. A place, person, picture, thing, concept, process result or data are some of the examples of entities. An entity may be *concrete* (a person or a book, for example) or *abstract* (like a holiday or a concept). An entity has a set of properties.

- ***Entity set***

Entity set is a set of entities of the same type that share the same properties. For example, the set of all employees can be defined as the entity set *Employee*.

***Weak entity set*:** If an entity set does not have sufficient attributes to from a primary key; such entity set is called weak entity set.

***Strong entity set*:** An entity set which has a primary key is called as strong entity set.

- ***Attributes***

Each entity has a set of specific characteristics which describe the object. These characteristic are called **attributes**. For example, attributes of *Employee* entity are *Emp-No*, *Emp-Name*, *Address*, etc.

Simple attribute: Attributes which are not divided into subparts (i.e. other attributes), are called *Simple attributes*.

Composite attribute: Some attributes can be divided into smaller subparts (i.e. other attributes). An attribute which is composed of several more basic attributes is called *composite attribute*. For example, *name* attribute has sub attributes *first-name*, *middle-name*, and *last-name*, thus, *name* is a composite attribute.

Single-valued attribute: The attributes have a single value for a particular entity, is called *single-valued attributes*.

Multi-valued attribute: If an attribute has a set of values for a specific entity, then it is said to be *multivalued*. Consider an employee entity set with the attribute *phone-number*. An employee may have zero, one, or several phone numbers, and different employees may have different numbers of phones.

- ***Relationship* (*Relationship set*)**

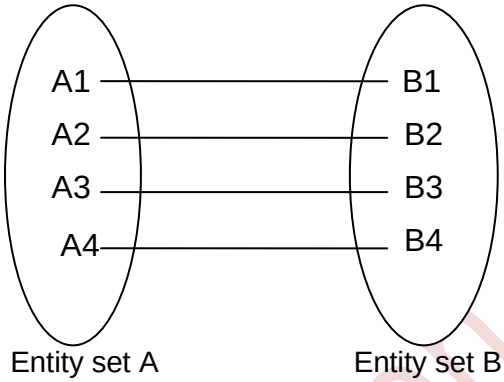
A *relationship* is an association among several entities. Relationship holds together the various components of E-R model. Relationships in databases are often ***binary***. Other types of relationships are ***ternary*** and ***recursive*** relationships.

Mapping cardinalities

Mapping cardinalities express the number of entities from one entity set associates with entities of another entity set by a binary relationship.

1. *One-to-one*

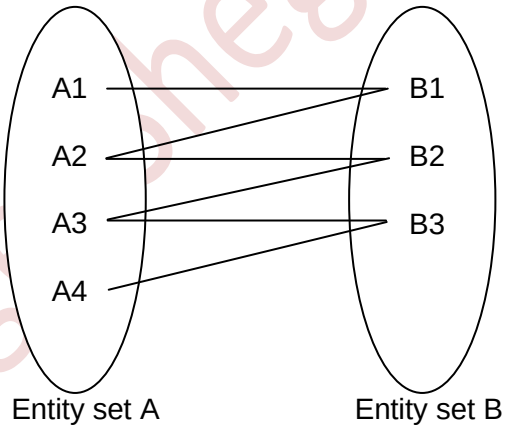
An entity in A is associated with at most one entity in B, and an entity in B is also associated with at most one entity in A. This is illustrated in Fig. 5.



[Fig. 5 : One-to-one relationship **]**

2. *Many-to-one*

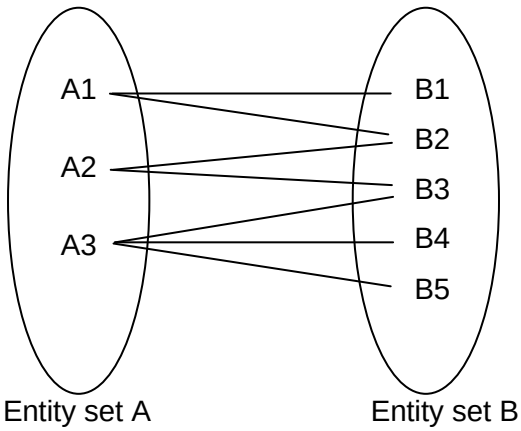
An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number of entities in A. This is illustrated in Fig.6.



[Fig. 6 : Many-to-one relationship **]**

3. *One-to-many*

An entity in A is associated with any number of entities in B. An entity in B. however, can be associated with at most one entity in A. As shown in Fig.7.



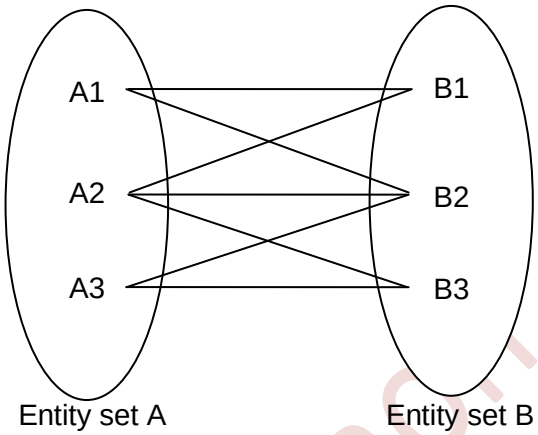
[Fig. 7 : One-to-many relationship **]**

NOTE:- *When there is a one-to-many relationship between A and B, there is a Many-to- one relationship between B and A.*

Prof. V. V. AGARKAR
D. M. Burungale Science & Arts College, Shegaon

4. ***Mane-to-many***

An entity in A is associated with any number of entities in B and an entity in B *is* also associated with any number of entities in A. This is illustrated in Fig.8.

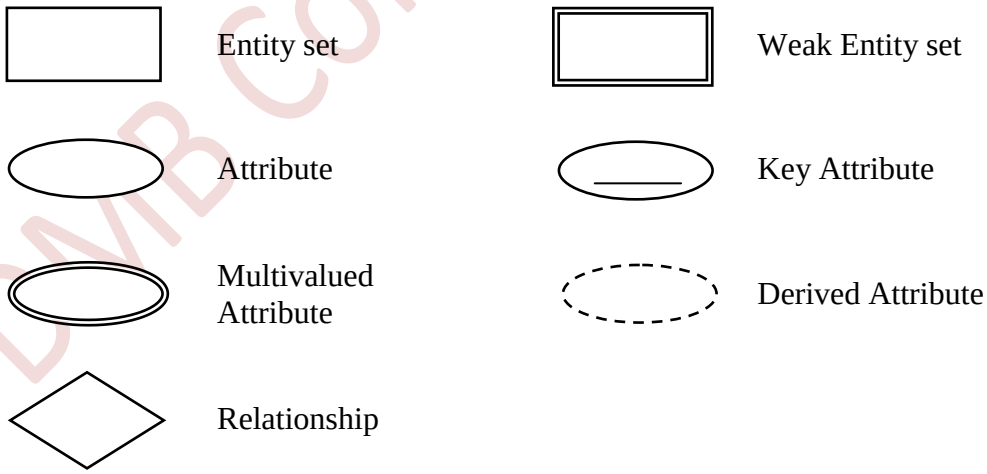


[Fig. 8 : Many-to-many relationship **]**

E-R Diagram

The overall logical structure of a database can be expressed graphically by an E-R diagram. An E-R diagram is a method of representing entities, attributes and relationships. An E-R diagram consists of the following components:

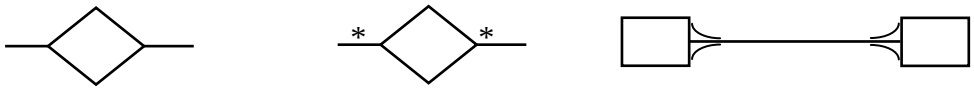
1. ***Rectangles***, which represent entity sets
2. ***Ellipses***, which represent attributes
3. ***Diamonds***, which represent relationship among entity sets
4. ***Lines***, which link attributes to entity sets and entity sets to relationships.



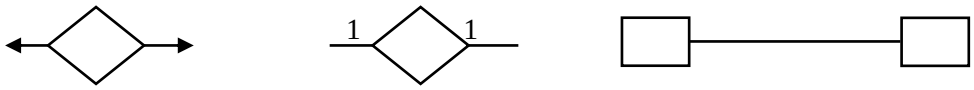
[Fig. 9: Symbols used in E-R diagrams **]**

When an E-R diagram is built, the first step is to define the entities. The next step is to define the relationship between entities. The final step is to identify the attributes that belong to each entity. *Attributes of an entity set that are members of the primary key are underlined*. ***Each component except lines is labeled with a unique name.***

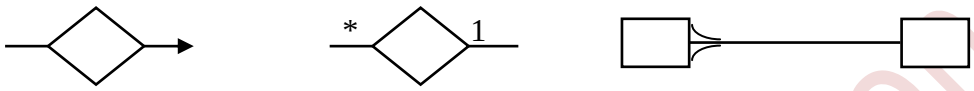
- ***Many-to-many Relationship***



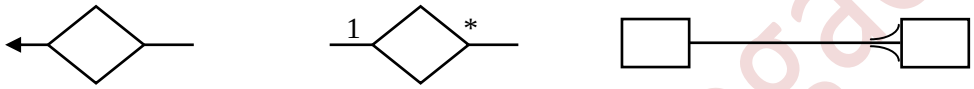
- ***One-to-one Relationship***



- ***Many-to-one Relationship***



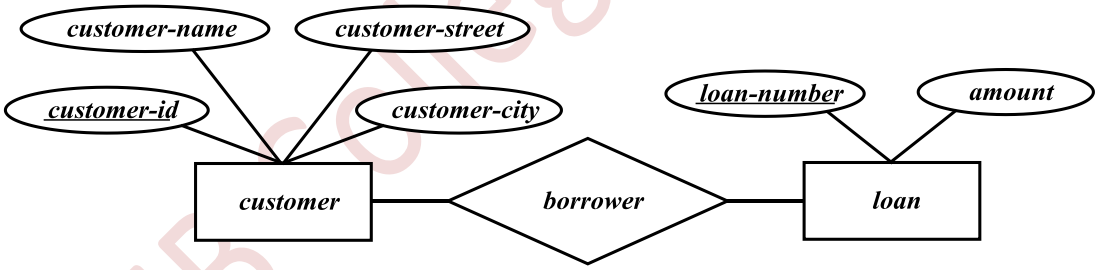
- ***One-to-many Relationship***



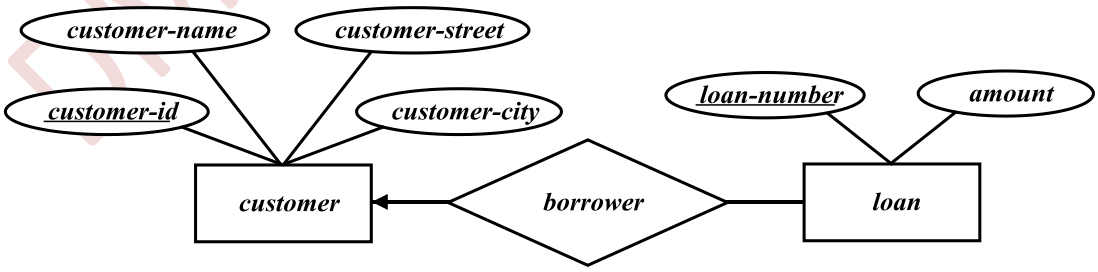
[Fig. 10 : Alternative symbols for Relationships used in E-R diagrams **]**

Examples:

1. Consider the following entity-relationship diagram in Figure 11, which consists of two entity sets, *customer* and *loan*, related through a binary relationship set *borrower*. The attributes associated with *customer* are *customer-id*, *customer-name*, *customer-street*, and *customer-city*. The attributes associated with *loan* are *loan-number* and *amount*.



(a) In above E-R diagram, the relationship set *borrower* is ***many-to-many.***

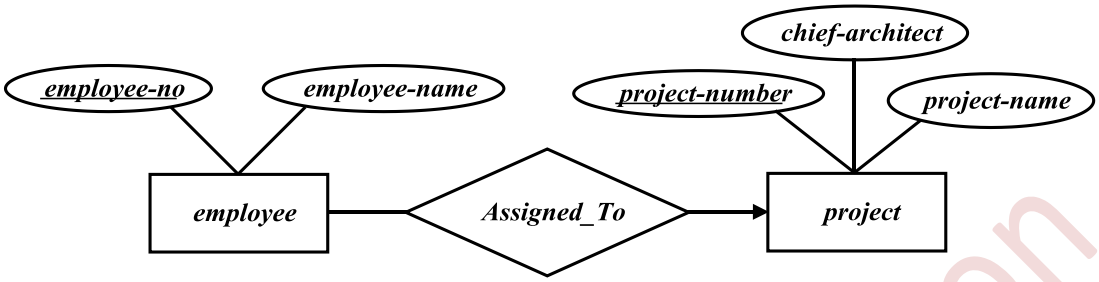


(*b*)

[Fig. 11 : E-R diagram corresponding to customers and loans**]**

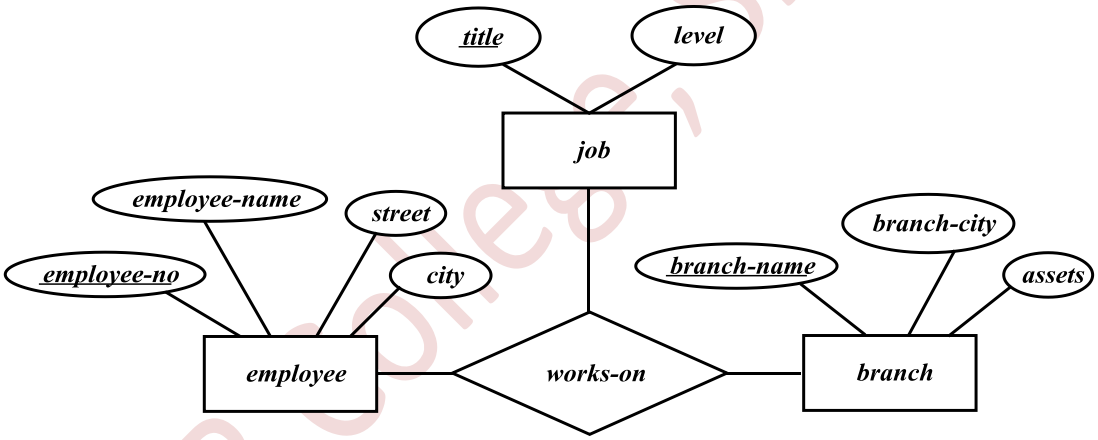
If the relationship set *borrower* were ***one-to-many***, from *customer* to *loan*, then the line from *borrower* to *customer* would be directed, with an arrow pointing to the *customer* entity set (Figure 11(*b*)).

2. Following is a E-R diagram shown in Fig. 12, which consists of two entity sets, *employee* and *project*, related through a relationship set *Assigned_To* ***(many-to-one)***. The attributes associated with employee are *name* and *number*. The attributes associated with project are *project-name*, *project-no* and *chief-architect*.



[Fig. 12 : E-R diagram corresponding to employee and project **]**

3. Following entity-relationship diagram in Figure 13, this consists of three entity sets, *employee*, *job*, and *branch*, related through the ***ternary relationship*** set *works-on*. The attributes associated with *employee* are *employee-id, employee-name*, *street*, and *city*. The attributes associated with *branch* are *branch-name* and *branch-city, assets*. The attributes associated with *job* are *title* and *level*.



[Fig. 13 : E-R diagram corresponding to employee, branch and job**]**

Reduction of an E-R diagrams to Tables

The E-R diagram can be represented by a collection of tables. For each entity set and for each relationship set in the E-R diagram, there is a unique table to which we assign the name of the corresponding entity set or relationship set. Following are the steps:

1. **Representation of Entity sets:**

   An entity set E with attributes a_1, a_2, a_3, . . . , a_n is represented by a table called E with n distinct columns, each of which corresponds to one of the attributes of the entity set E. Each row in this table corresponds to one entity of the entity set E.

2. **Representation of Relationship sets:**

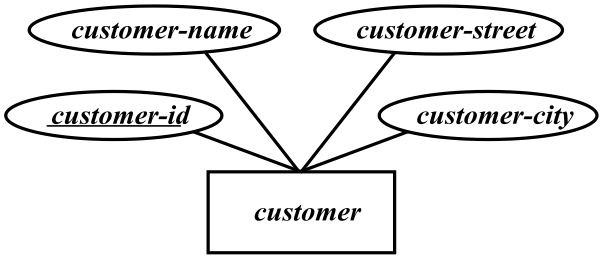
   If R is a relationship set. This relationship set is represented by a table called R with the primary key attributes of associated entity sets are taken as columns.

Prof. V. V. AGARKAR
D. M. Burungale Science & Arts College, Shegaon

Example 1: Reduce the E-R diagram shown in Fig. 11(a) into tables.

Answer:

1. The entity set **customer** is treated as table and its attributes: customer-id, customer-name, customer-street and customer-city are treated as columns of the table.

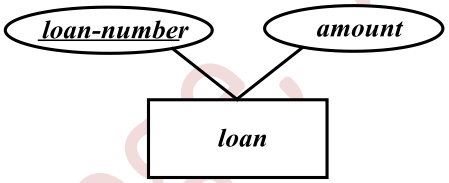


Thus CUSTOMER table is:

CUSTOMER(customer_id, customer_name, customer_street, customer_city)

The primary key of table CUSTOMER is customer_id.

2. The entity set **loan** is treated as table and its attributes loan-number and amount are treated as columns of the table.



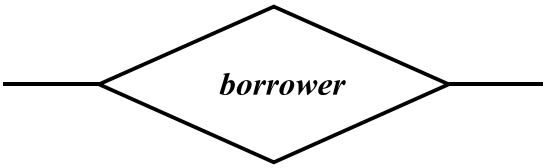
Thus LOAN table is:

LOAN(loan_number, amount)

The primary key of table LOAN is loan_number.

3. Relationship set **borrower** will be treated as table in relational model. This relationship set involves the two entity sets: *customer* and *loan*.

- *customer*, with the primary key *customer-id*
- *loan*, with the primary key *loan-number*



Thus BORROWER table is:

BORROWER(customer_id, loan_number)

The primary key of table BORROWER is a combination of customer_id and loan_number.

Thus, E-R diagram in Fig. 11 (*a*) is represented in relational model as follows:

```
CUSTOMER (customer_id, customer_name,
          customer_street, customer_city)
LOAN (loan_number, amount)
BORROWER (customer_id, loan_number)
```

Example 2: Reduce the E-R diagram shown in Fig. 12 into tables.

Answer:

Apply the steps as shown in above example (example 1).

E-R diagram in Fig. 12 is represented in relational model as follows:

```
EMPLOYEE(employee_no, employee_name)
PROJECT(project_number, project_name, chief_architect)
ASSIGNED_TO (employee_no, project_number)
```

Example 3: Reduce the E-R diagram shown in Fig. 13 into tables.

Answer:

Apply the steps as shown in above example (example 1).

E-R diagram in Fig. 13 is represented in relational model as follows:

```
EMPLOYEE (employee_no, employee_name, street, city)
BRANCH (branch_name, branch_city, assets)
JOB (title, level)
WORKS_ON (employee_no, branch_name, title)
```

Functional Dependency (FD)

A functional dependency is a concept in database management that describes the relationship between two attributes or set of attributes within a relation (table). In a table, attributes (columns) are functionally dependent on one another; if the value of one attribute uniquely determines the value of another attribute. Functional dependencies are crucial for database design and normalization processes as they help ensure data integrity and eliminate redundancy.

The Fundamental definition of functional dependency (FD) is

> ***Given a relation R, attribute Y of R is functionally dependent on attribute X of R i.e. X → Y, if and only if each X-value in R has associated with precisely one Y- value in R.***

The alternative definition of FD is:

> ***Given a relation R, attribute Y of R is functionally dependent on attribute X of R, if and only if, whenever two tuples of R agree on their X-value, they also agree on their Y-value.***
>
> ***That is for any two tuples t_1 and t_2 of R, if $t_1[x] = t_2[x]$, implies $t_1[y] = t_2[y]$.***

- ***Notations of functional dependency***

Functional Dependency is represented in the form of an equation. Here, you have a set of attributes (A, B, C, etc.) and an arrow (→) denoting the Dependency. Suppose ***X*** and ***Y*** are two attributes in a relation R, then the functional dependency is shown as:

$$X \rightarrow Y$$

The meaning of this notation is:

- "***X***" determines "***Y***"
- "***Y***" is functionally dependent on "***X***"
- "***X***" is called determinant
  "***Y***" is called object of the determinant

Example:- Consider a relation/table called "Students" with attributes (columns) "StudentID", "Name", "DOB", and "Department".

StudentID	Name	DOB	Department
101	Smith	22-05-2004	Chemistry
102	John	14-04-2004	Mathematics
103	Blake	28-02-2004	Chemistry

Each student has a unique ID, in this case, the functional dependency would be:

StudentID → Name
StudentID → DOB
StudentID → Department
Name, DOB → Department

This means that given a StudentID, you can uniquely determine the associated department. Similarly, given a combination of Name and DOB, you can also uniquely determine the associated department.

However, it's important to note that the reverse is not necessarily true. That is, given a department, you cannot necessarily determine the StudentID or the combination of Name and DOB uniquely.

Normalization

Normalization is a process used in database design to organize data into tables in such a way that redundancy and dependency are minimized. It involves breaking down a table into smaller tables. Normalization helps in reducing data anomalies such as – insertion, update, and deletion anomalies.

Normalization is a multi-step process, which consists of several steps and each step referred to as a ***Normal Form***. Basically, normalization is an iterative process, meaning you might need to apply multiple normal forms to achieve the desired level of data integrity and efficiency. However, it's also essential to balance normalization with performance considerations. Over-normalization can lead to complex queries and slower performance. The normal forms after each step is called ***first***, ***second***, ***third***, ***fourth*** normal forms.

- ***Objectives of Normalization***

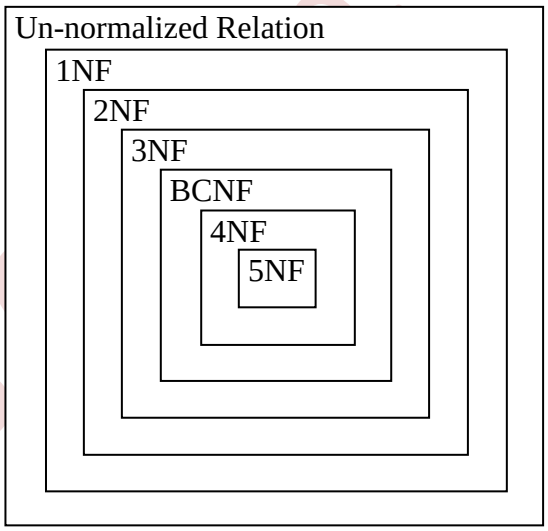
The objectives of the normalization process are:

1. To make it possible to represent any relation in the database.
2. To free relations from undesirable insertion, updation and deletion anomalies.
3. Enforce Integrity Constrains.
4. To increase data consistency, minimal data redundancy and maximum stability.

Normal Form

A relation is said to be in a particular normal form if it satisfies a certain specified set of constraints. *Dr. Edgar F. Codd* introduced the concept of normalization. Codd originally defined first, second and third forms (1NF, 2NF, 3NF). Codd's original definition of 3NF suffered from certain inadequacies. A revised stronger definition was given by Boyce and Codd that was the new definition of 3NF sometimes called Boyce/Codd Normal Form (BCNF). Normal forms that have been identified are:

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Fourth Normal Form (4NF)
- Boyce-Codd Normal Form (BCNF)



[Fig. 14 : Levels of Normalization]

Each normal form contains in it the previous normal form(s). For example, If any relation which is in 3NF is also a 2NF and 1NF relations. Informally, a relational database table is often described as "***normalized***" if it meets **3NF**. Most 3NF tables are free of insertion, deletion and update anomalies.

- ***Unnormalized Relation***

A relation is called ***unnormalized*** relation, if each tuple (row) may contains multiple set of values for some of the attribute (column), these multiple values in a single row are also called *non-atomic* values.

Let us consider a **LIBRARY_ISSUE** table as shown below:

Mem_ID	Mem_Name	Contact	Book_code	Cat_code	Date	
					Issue	Return
M001	Sachin	4142319	B0020	Science	18/03/98	18/04/98
M008	Rahul	8600909	B0189	Arts	18/03/98	18/04/98
			B0090		18/03/98	18/04/98
M067	Anil	4219498	B0656	Political	19/03/98	98/04/98
M123	Saurav	5670967	B0198	Classic	21/03/98	21/04/98
			B0212		21/03/98	21/04/98
			B0400		21/03/98	21/04/98
M880	Ajay	3289656	B0001	Economics	22/03/98	22/04/98

[Table 1 : Unnormalized LIBRARY_ISSUE Relation **]**

In the above relation (Table-1), **Date** column contains set of values i.e. non-atomic values and hence it is unnormalized relation.

First Normal Form (1NF)

A relation is said to be in First Normal Form (or 1NF) if the values in the domain of each attribute are atomic; that is they cannot be decomposed into component values.

Example :- Consider Table-1 LIBRARY_ISSUE, it is unnormalized relation. To convert it into 1NF, remove the non-atomic values; make the attribute with atomic values. A key that will uniquely identify each record should be assigned to the relation. Here the primary key is a combination of Mem_ID and Book_code.

The normalized form of Table-1 is given below in Table-2:

Mem_ID	Mem_Name	Contact	Book_code	Cat_code	Issue_Date	Return_date
M001	Sachin	4142319	B0020	Science	18/03/98	18/04/98
M008	Rahul	8600909	B0189	Arts	18/03/98	18/04/98
M008	Rahul	8600909	B0090	Arts	18/03/98	18/04/98
M067	Anil	4219498	B0656	Political	19/03/98	98/04/98
M123	Saurav	5670967	B0198	Classic	21/03/98	21/04/98
M123	Saurav	5670967	B0212	Classic	21/03/98	21/04/98
M123	Saurav	5670967	B0400	Classic	21/03/98	21/04/98
M880	Ajay	3289656	B0001	Economics	22/03/98	22/04/98

[Table 2 : LIBRARY_ISSUE Relation in 1NF Form **]**

Second Normal Form (2NF)

A relation R is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key and not on a part of the primary key.

Example :- Consider following table **ADMISSION** containing courses that are taken in a certain semester, and it contains the following data:

|---Primary Key ---|

CourseID	Semester	#Places	CourseName
IT101	2009-1	100	Programming
IT101	2009-2	100	Programming
IT102	2009-1	200	Databases
IT102	2009-1	150	Databases
IT103	2009-2	120	Web Design

[Table 3 : ADMISSION Relation not in 2NF Form **]**

In the above relation ADMISSION *CourseID+Semester* is a primary key and the functional dependencies are:

CourseID, Semester → #Places
CourseID → CourseName

This is **not in 2NF**, because the fourth column CourseName does not fully dependent upon the entire key CourseID+Semester - but only on a CourseID i.e. part of it. Thus, we have duplicate information - several rows telling us that IT101 is programming, and IT102 is Databases. So fix this by converting ADMISSION relation into 2NF.

To convert into 2NF remove CourseName from ADMISSION table and put it into new table called COURSE, where CourseID is the entire key. Now there is no redundancy.

ADMISSION

|-Primary Key -|

CourseID	Semester	#Places
IT101	2009-1	100
IT101	2009-2	100
IT102	2009-1	200
IT102	2009-1	150
IT103	2009-2	120

COURSE

Primary Key

CourseID	CourseName
IT101	Programming
IT102	Databases
IT103	Web Design

[Table 4 : ADMISSION Relation in 2NF Form **]**

Third Normal Form (3NF)

A relation R is in third normal form (3NF) if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key.

An attribute C ***is transitively dependent*** *on attribute A if there exists an attribute B such that: A → B and B → C.*

Example :- Consider following relation **ADMISSION** containing courses that are taken in a certain semester and the respective teachers for these courses:

|--Primary Key --|

CourseID	Semester	#Places	TeacherID	TeacherName
IT101	2009-1	100	332	Mr Jones
IT101	2009-2	100	332	Mr Jones
IT102	2009-1	200	495	Mr Bentley
IT102	2009-1	150	332	Mr Jones
IT103	2009-2	120	242	Mr Smith

[Table 5 : ADMISSION Relation not in 3NF Form **]**

In the above relation ADMISSION *CourseID+Semester* is a primary key and the functional dependencies are:

f1: CourseID, Semester → #Places
f2: CourseID, Semester → TeacherID
f3: TeacherID → TeacherName

Now, by FDs *f2* and *f3*, the TeacherName is dependent on TeacherID and TeacherID dependent on primary key - so *attribute TeacherName is transitively dependent on primary key*, thus, this is **not in 3NF**. To convert above relation into 3NF- take TeacherName out of this table, and put it in new table called TEACHER, which has TeacherID as the key.

ADMISSION

|-Primary Key -|

CourseID	Semester	#Places	TeacherID
IT101	2009-1	100	332
IT101	2009-2	100	332
IT102	2009-1	200	495
IT102	2009-1	150	332
IT103	2009-2	120	242

TEACHER

Primary Key

TeacherID	TeacherName
332	Mr Jones
495	Mr Bentley
242	Mr Smith

[Table 6 : ADMISSION Relation in 3NF Form **]**

Boyce-Codd Normal Form (BCNF)

Don't write this in the answer

When a relation has more than one candidate key, anomalies may result even though the relation is in 3NF. 3NF does not deal satisfactorily with the case of a relation with overlapping candidate keys (i.e. composite candidate keys with at least one attribute in common). Hence, new definition for 3NF i.e. required.

The new definition is due to Boyce and Codd; hence the term *"Boyce/Codd Normal Form" (BCNF)* is often used to distinguish the new 3NF form the old.

The definition of BCNF is:

A relation R is in Boyce-Codd normal form (BCNF) if for every nontrivial functional dependency X→A, X is a super key. In other words, a relation is in BCNF if and only if, every determinant is a candidate key.

BCNF is based on the concept of a *determinant*. A ***determinant*** is any attribute (simple or composite) on which some other attribute is fully functionally dependent.

Example :- *Converting a Relation to BCNF*

Consider the relation ADDRESS which has three attributes STREET, CITY, and ZIP (Pin code).

```
ADDRESS (STREET, CITY, ZIP)
```

From the relation ADDRESS we have functional dependencies:

```
CITY, STREET  →  ZIP
ZIP  →  CITY
```

The relation ADDRESS is not in BCNF, *because ZIP is not a superkey*.

The relation ADDRESS has insertion anomaly, that is a city of ZIP code cannot be stored if the street is not given. To overcome this insertion anomaly, the relation ADDRESS has to be converted into BCNF. To do this, split relation ADDRESS into two relations R1 and R2. The relation R1 has two attributes STREET, ZIP, and the relation R2 has two attributes ZIP, CITY.

```
R1(STREET, ZIP)  and
R2(ZIP, CITY)
```

The splitting of the relation ADDRESS into two relations R1 and R2 eliminates insertion anomaly.

Fourth Normal Form (4NF)

The definition of 4NF is:

A relation R is in fourth normal form (4NF) if and only if it is in BCNF and, whenever there exists an multivalued dependency in R (for example X→Y), at least one of the following holds:

The multivalued dependency is trivial or

X is a super key for relation R.

Example 9: *Converting a Relation to Fourth Normal Form*

Consider the relation EMPLOYEE with the attributes employee number (ENO), project name (PNAME), and department name (DNAME) as shown below:

```
EMPLOYEE (ENO, PNAME, DNAME)
```

The relation EMPLOYEE has the following multivalued dependencies:

```
ENO  →  PNAME    (One employee can work in several projects)
ENO  →  DNAME
```

ENO is not the superkey of the relation EMPLOYEE. To convert the relation to fourth normal form decompose EMPLOYEE relation into two relations EMP-PROJ and EMP-DEPT as shown below.

```
EMP_PROJ(ENO, PNAME) and EMP_DEPT(ENO, DNAME)
```

Now the relations EMP-PROJ and EMP-DEPT are in fourth normal form.

■ ■ ■ ■ ■