UNIT III

DATABASE DESIGN AND THE E-R MODEL

Overview of the Design Process:

Design Process

In design process, the requirements of the users plays significant role. Since development of a database application is very different task, it includes database schema design. This design enables a user to design a security scheme in order to control access to data and design a database program that allows user to update and access the data within the database.

A complete database design can be created depending on database application environment. This database is created in such a way that it fulfills the requirements of the organization thereby solving multiple organizational issues.

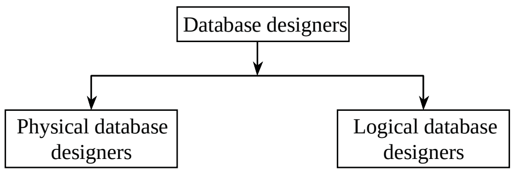
Design Phases

Design phase is a designing process that enables a designer to design a database based on needs of the application. This phase allows designer to directly decide the relations which are to be created, their attributes and constraints on the relations for small size applications. Such type of direct design process is complex for the real-world application, it is impossible for single person to understand the entire data requirements of an application. This is because, the design of the database should collaborate with application requirements, implement them in a high-level form which can be understood by the users of application and then transform the requirements into lower levels of the design. A high-level data model helps the database designer to create a conceptual framework that systematically arranges the data depending on the needs of the database users and a database structure satisfying the desired needs.

1. The first phase of database design is to completely characterize the data based on requirements of the database users. To complete such task, the database designer must collaborate with experts and users. The result of this phase will be a definition of user needs.
2. In the next phase, the designer must select the data model by utilizing its concepts and then translating their needs into a conceptual database schema. At this conceptual design phase, the developed schema gives a complete overview of the organization. To represent the conceptual design, the entity-relationship model is used. The conceptual schema defines the database entities, the attributes of entities, the relationships among the entities and constraints based on the entities and relationships. The conceptual design phase leads to the creation of an entity-relationship diagram which gives a graphical representation of the schema.

   The schema is reviewed by the designer to verify the data needs that can be fulfilled. The design should then try to remove all the redundant features that exist within the database schema. In addition to this, the designer must focus on storage information instead of describing the data and their relationships.
3. A completely developed conceptual schema must represent the functional needs of the organization. This enables a database designer to define a variety of operations which implements the data operations such as data searching, data modification, data updation, data retrieval and data deletion etc. Based on this conceptual design, the schema is reviewed by the designer ensuring the functional needs.

Designers of Database



The responsibility of database designer is to identify,

(i) Data to be stored

(ii) The structure of the data.

Database designer interact with all the users of database so as to identify the requirements of database. Each designer interacts with a group of users and develops a view. All the views are then analyzed in order to develop a final view of the database that satisfies the requirements of all the users.

Database designers are classified into two types. They are,

(i) Logical database designers

(ii) Physical database designers.

Logical database designers are the one who identify the data and the relationship among the data. Physical database designer decides how to represent the logical database design physically.

(i) Logical-design Phase

In this phase, the designer designs a high-level conceptual schema to implement the data model with in the database system. The designed data is completely different from relational data model having conceptual schema specified using the entity relationship model into a relation schema.

(ii) Physical-design Phase

It defines the physical features of the database. These types of features involve the type of file organization and selection of index structures.

DESIGN ALTERNATIVES

Database design process specifies the way of representing the database design in various types of alternatives like places, people and differently recognizable things. For instance, in the university database, entities are guides, students, courses and departments. The different entities are associated to one another in different ways which is required to be occupied in the database design.

To design a database schema, the database must not include the following,

1. Redundancy

It is a process that includes multiple copies of data. For instance, in a database course the course id and course title are stored with every offered course. Here, the course title will be stored redundantly with every course offered. It will store only the course id with every course offered and relate the title with the course-id at once in course entity.

A relational schema named "university" may contain a relation with section information and an individual relation with course information. Assume that, the user has a single relation where all information of the course such as course title, course-id, and dept-name credits once for every course offered. It is clear that, the information about courses will be stored redundantly.

The major problem for such of redundant information is the duplication of information within the database. This makes data representation difficult if the information is updated without any precautionary measures. For instance, different courses offered may contain same course id but contains distinct titles. Later, it can create confusion about identifying correct course title. So, the information should exist only at one place.

2. Incompleteness

Incomplete data is also one of the major drawbacks of database design. For instance, assume that, the user contains entities similar to the course offered without an entity similar to courses. Similarly, as in relations, assume that a single relation repeats all information of the courses at once for every part of the course offered. Later, it will be impossible to represent information based on a new course, until that course is offered.

The problem with this design is that it stores the null values for the selection information. Such information can be avoided by primary-key constraints. Thus, creation of such database design must be avoided.

THE ENTITY-RELATIONSHIP MODEL (E-R MODEL)

Data Model

Data Model is a mean of modeling data i.e., to give a shape to the data (or) to give a figure to the stored data. A data model makes easier to understand the meaning of the data by its figure and ensures that the users understand,

- ❖ The nature of the data without requiring any physical implementation details
- ❖ The view of data from other user's perspective
- ❖ The use of data across application areas.

Types of Data Model

The data models are divided into three different groups, which are as follows,

(a) Object-based logical models

(b) Record-based logical models

(c) Physical data models.

(a) Object-based Logical Models

Object-based logical models are used in describing data at logical-level and view-level. Logical level is used by DBA, so as to make a decision regarding what data are to be stored in the database and what relationships must exist among those data. On the other hand, view level describes only part of the entire database that is to be viewed by the database user i.e., it hides the details of the information stored.

The different types of object-based logical models include,

(i) The entity-relationship model

(ii) The object-oriented model

(iii) The semantic data model

(iv) The functional data model.

(i) The Entity-Relationship Model

An entity is a real-world object that can be uniquely identified. The Entity-Relationship Model (E-R Model) is based on a collection of basic objects called entities and the relationship among these objects. Basically, E-R diagram is the graphical representation of the entire logical structure of database.

(ii) The Object-oriented Model

Like the E-R model, the object-oriented model is based on a collection of objects. An object contains values stored in instance variables, methods (bodies of code) that operate on the object.

Objects containing similar types of values and similar methods are combined to form classes. A class defines objects. The only way in which one object can access the data of another object is by invoking a method of that other object. This action is called sending a message to the object.

(iii) The Semantic Data Model

A semantic data model is a more high-level data model that makes it easier for a user to give starting description of data in an enterprise. These models contain a wide variety of relations that helps to describe a real application scenario. A DBMS cannot support all these relations directly, so it is built only with few relations known as relational model in DBMS.

(iv) The Functional Data Model

The functional data model makes it easier to define functions and call them during data processing.

(b) Record-based Logical Model

Record-based logical model describes data at logical and view levels. It stores the data in the form of records (documents) of several types. Each record has fixed number of (fields or) attributes and each field contains fixed length.

In comparison to object-based data model, the record- based logical model describes the overall logical structure of the database with higher-level implementation.

The record-based models are of three types,

(i) Relational model

(ii) Network model

(iii) Hierarchical model.

(i) Relational Model

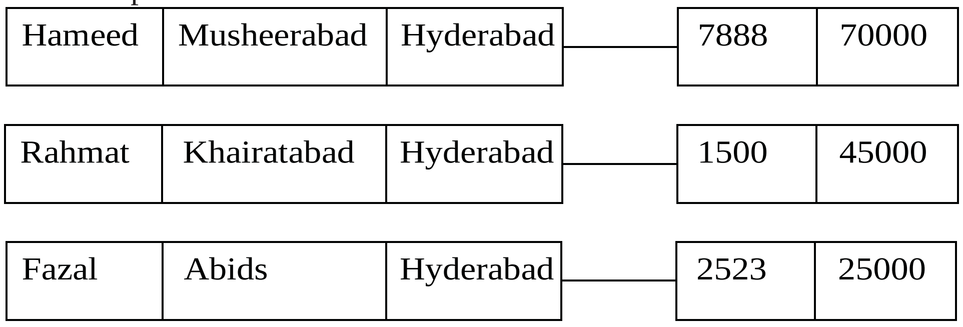
The relational model represents both data (entities) and relationships among data in the form of tables. Each table has multiple columns with a unique name. Consider the following relational model.

Example

Customer Relation			
Customer-name	**Street**	**City**	**Acc. No.**
Hameed	Musheerabad	Hyderabad	7888
Rahmat	Khairatabad	Hyderabad	1500
Fazal	Abids	Hyderabad	2523

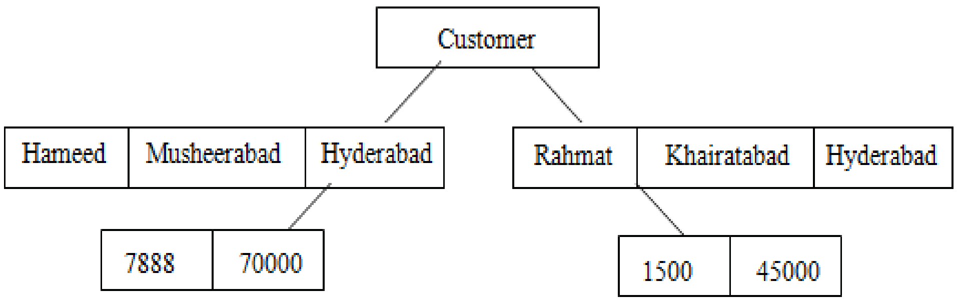
(ii) Network Model

Network model represent data in the form of collection of records and relationships among data are represented by links. The links can be viewed as pointers.



(iii) Hierarchical Model

Hierarchical model is same as the network model i.e., data in the hierarchical model is represented as a collection of records and relationship among data and are connected by links. The links can be viewed as pointers. But, the difference between these two models is that in network model the records in the database are represented in the form of graphs, whereas in hierarchal model, they are represented in the form of trees.



(c) Physical Data Models

Physical data models give the description of data at the lowest level. It describes about the storage of data in low-level data structures.

Two types of physical data models are,

- Unifying model
- Frame-memory model.

<u>ER Model</u>

Entities

An entity is a real-world object that can be uniquely identified. There are two types of entities.

(i) Strong entity

(ii) Weak entity.

(i) Strong Entity

Strong entity is the one that does not depend on other entities.

For example, a chairman of a company does not depend on anyone for final decisions. Hence, chairman is strong entity.

(ii) Weak Entity

Weak entity is the one that depends on other entities for existence.

For example, if an employee is retired then we do not need to store the details of his dependents. Hence, the dependents (children) entity is weak entity.

Relationships

Relationship is an association among various entities. The entities that take part in a relationship are called “participants” and the number of participants of a given relationship is called “degree” of relationship.

If every instance of an entity participates in at least one instance of a relationship, then the participation is said to be total, else, it is said to be partial participation.

A relationship can be one-to-one, one-to-many or many-to-many.

Attributes:

Each entity and relationship has a property called attributes. These attributes can be,

(i) Simple Attributes

These attributes are also called as atomic attributes which cannot be subdivided further. For example, the attributes like roll number and class of ‘student’ entity cannot be further subdivided.

(ii) Composite Attributes

An attribute which can be further divided into smaller components are called composite attributes.

For example, the attribute name can further be divided into first name, middle name and last name. Similarly, the attribute address can further be divided into house number, city, street, country.

(iii) Single-valued Attributes

Certain attributes take only a single value in all instances. For example, the age of a person is a single-valued attribute as man cannot have two age.

(iv) Multi-valued Attributes

Attributes that can have more than one value at a time for an instance are called multi-valued attributes. For example, the color of the product. A product might be multicolored in this case it takes more than one value at a time.

(v) Stored and Derived Attributes

Some attributes need not be stored, but can be derived from available other attributes.

For example, the total number of students in a class can be calculated by counting the number of student records. Similarly, the age of a student can be calculated, by subtracting the date-of-birth field from present date. The age field is called derived attribute and date-of-birth is called stored attribute.

Entity Sets

An entity set is defined as a group of entities that have similar types or attributes. For instance, employees working in an organization are defined as entities E_1, E_2, E_3, E_4, ..., which may contain similar attributes defined under a specific entity type called

‘Employee’. Here, the group of entities i.e., {E_1, E_2, E_3, E_4, ...} is referred to as an entity set.

Every attribute of the entity set has a set of pairs defined by (attribute, data value). This will help in describing the characteristic or the nature of the entity. Example, an ‘employee’ entity is defined by a set of pair as, {(Emp_name, John), (Emp_no, 0023), (Emp_street, NorthMoody), (Emp_city, Illionis)}.

If an entity set contains enough attributes for creating a primary key, then it is termed as strong entity set. On the other hand, if an entity set does not contain enough attributes for creating a primary key then it is termed as weak entity set. Typically, member corresponding to a strong entity set is called dominant entity whereas member corresponding to a weak entity set is called subordinate entity.

Relationship Sets

A collection of similar relationships is called a relationship set and is denoted by a rhombus.

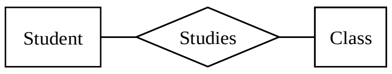


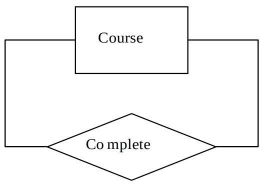
Figure: Student-class Relationship

Degree

The number of entities associated with a relationship set is known as degree of ntities.

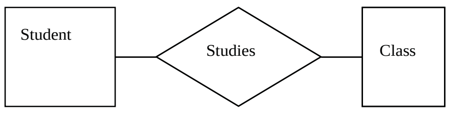
Unary Relationship

In a unary relationship the association is within a single entity.



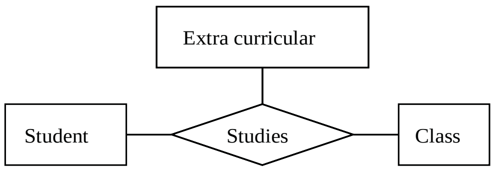
For example, it is not possible to opt for JAVA until the course in ‘C’ is being finished and for J2EE one should have completed JAVA. This kind of relationships is shown by unary associations.

Binary Relationship



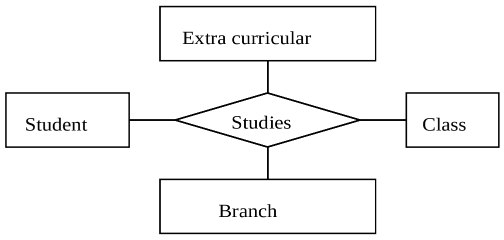
A relationship that associates two entities is known as binary relationship.

Ternary Relationship



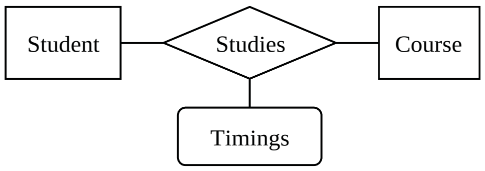
A relationship that associates three entities is known as ternary relationship.

Quaternary Relationship



A relationship that associates with four entities is known as quaternary relationship.

A relationship can also have an attribute called a descriptive attribute.



For example, student takes a course at an institution. Then the relationship among the two entities i.e., student and course is 'studies'. Each student can choose from allowable time schedules. Hence timing can be considered as an attribute of relationship 'studies'.

Role:

The function of the entity set in the relationship is called 'role'. Specifying roles is not always required, as it is implicit. However in special relationship like recursive relationship, where an entity participates in a relationship with different roles.

In the above ER diagram, it can be easily interpreted that employee who is a manager works for an organization. If the role is not specified, it is not possible to know the post of the employee.

Constraints:

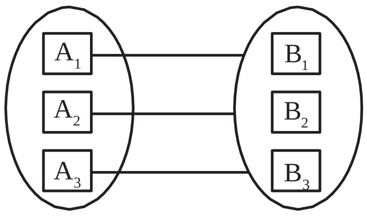
Mapping Cardinalities

Mapping cardinalities describe the relationship sets existing within the database. It is a technique that specifies the relationship sets which include more than two entity sets. Mapping cardinalities also called as cardinality ratios show the number of entities that can be related to another entity through a relationship set. In this, the user should focus on binary relationship sets.

The mapping cardinality for a binary relationship set R between entity sets A and B should follow one of the relations,

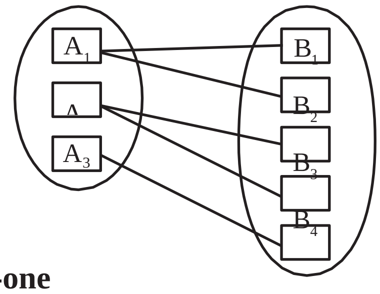
(a) One-to-one

In this, an entity in section A can be related to single entity of section B and entity of section B is related to single entity of section A.



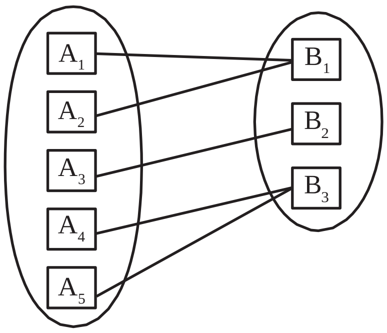
(b) One-to-many

In this relationship, an entity of section A is related with zero or more entities of section B. An entity of section B can be related with only a single entity of section A.



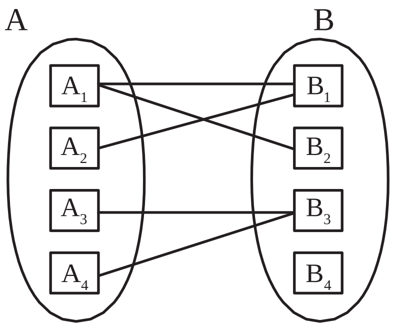
(c) **Many-to-one**

In this relationship, an entity of section A is related with only a single entity of section B and entity of section B can be related with zero or more entities of section A.



(d) **Many-to-many**

In this relationship, entity is related with zero or more entities of section B and entity of section B is related with zero or more entities of section A.



Mapping cardinality for a specific relationship set completely depends on the real time relationship set that is modeled.

Significance of keys:

A key for an entity is a set of attributes that differentiate set of entities from each other. Keys are also used to identify relationships among each other. The concepts of super key, candidate key and primary key can be applied to entity sets as they are applied to relation schemas.

The corresponding notions of keys for relationships can be defined as,

The primary key of an entity set differentiates the different entities of the set. Similar mechanism is required to differentiate between the different relationships of a relationship set.

Let primary-key (E_i) represents the set of attributes that makes the primary key for entity set E_i. Let R represents a relationship set including entity sets E_1, E_2, E_3, E_4, ... E_n. Consider that the attribute names of all primary keys are unique. The primary key composition for a relationship set based on the set of attributes related with the relationship set R.

When the relationship set R doesn't have attributes related with it then the set of attributes will be,

primary-key (*E1*)□□ primary-key (*E2*) □primary-key(E3)….. primary-key(En)

This shows an individual relationship set R.

When the relationship set R contains attributes such as a_1, a_2, a_3, a_4, ... a_m related with it then the set of attributes will be,
primary-key (*E1*) □ primary-key (*E 2*) □ primary-key(E3)….. primary-key(En)□□□ (a_1, a_2, a_3, a_4, ... a_m).
This shows an individual relationship in set R.

In the above cases, both the set of attributes primary-key (*E1*) □ primary-key (*E2*) □ ….□ primary-key(En) makes a super key for the relationship set.

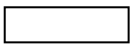
When the attribute names of primary keys are not individual throughout entity sets then the attributes are renamed to differentiate them.

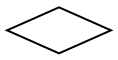
When an entity set takes part in a relationship set more than once then the role name is used rather than the entity set name to obtain an individual attribute name. The primary key structure for the relationship set is based on the mapping cardinality of the relationship set.

Entity-Relationship Diagrams

An entity is a real-world object that can be uniquely identified. The Entity-Relationship Model (E-R Model) is based on a collection of basic objects called entities and the relationship among these objects. Basically, E-R diagram is the graphical representation of the entire logical structure of database.

Symbols Used in E-R Model

Rectangle →  → Rectangle represents entities

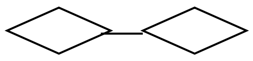
Diamond →  → Diamond represents relationship among entities

Ellipse → → Ellipse represents attributes (characteristics of entities)

Line → ——— → Lines represent link of attributes to entities to relationships.

Dashed lines → ------ → It links the attributes of one relationship set with another.

Double lines → ═══ → it shows the overall participation of an entity in a relationship set.

Double diamonds →  It connects strong relationship set to weak entity sets.

Consider the below ER-diagram,

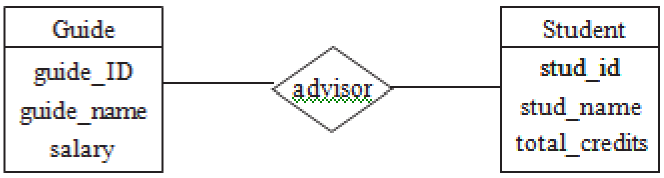


Figure (i): ER-diagram Relating to Guides and Students

The above diagram contains two entity sets such as guide and student associated through a binary relationship set 'advisor'. The attributes related with 'guide' are guide_ID, guide_name and salary. The attributes related with 'student' are stud_id, stud_name and total_credit.

When a relationship set contains related attributes then such attributes are enclosed in a rectangle and are connected with a dashed line to the diamond which represents the relationship set.

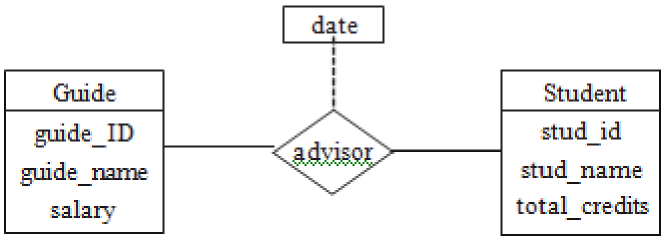
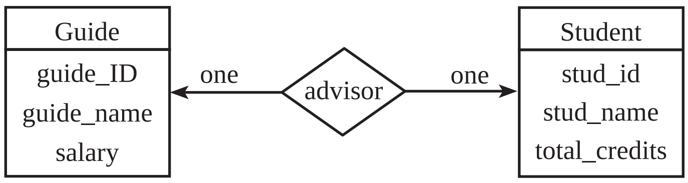


Figure (ii): Relationship Set

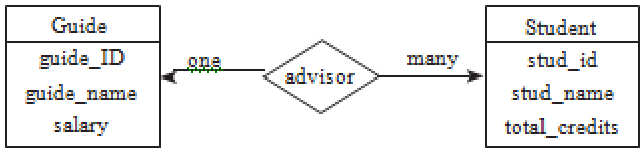
Figure: Relationship Set

The above diagram shows that the 'date' descriptive attribute connected to the relationship set advisor defining the data on which a guide becomes advisor. The various types of relationships existing between the above E-R diagram is depicted below,

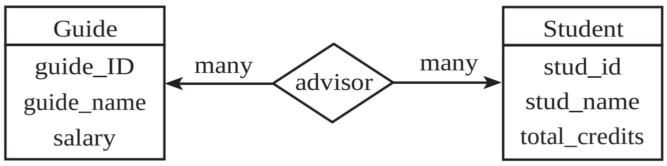
(a) One-to-one relationship



(a) one-to-many relationship



(b) Many-to-many relationship



Weak Entity Sets

A condition wherein a key attribute of one entity set belongs to another entity set is called weak entity set.

Principal Sources of Weak Entity Sets

(a) The primary source of weak entity set is that the entity sets possess atleast one many-to-one relationship with set E_1 are submits of entities present in the entity set E_2, then there is a possibility of the entities of E_2 to possess the names present in E_1. Hence, the names present in the entity set E_1 cannot be termed as unique unless the names present in the subordinate entity set E_2 are not taken into account.

Example

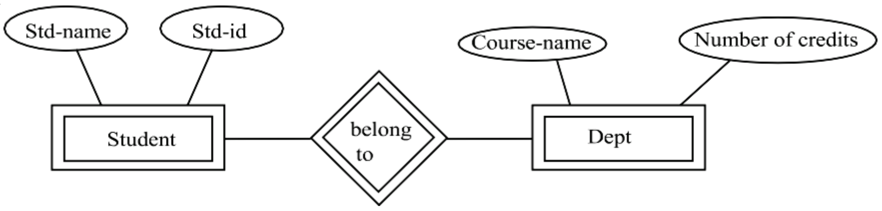


Figure: Weak Entity Set

In the above example the attributes std-name cannot be consider as the primary key since there is a possibility of one or more students to have the same name. However, even the attributes ‘std-id’ cannot be designated as a key since there is another possibility of using same Id’s in different courses. Hence, ‘student’ forms a weak entity set and the attributes ‘std-id’ and ‘course-name’ the keys of the weak entity set.

(b) The second source of weak entity set is that the entity sets sometimes does not have attributes of their own (as in multi-way relationship). In such cases, the entity sets get their keys from the key attributes of the connecting entity sets.

ER DIAGRAM FOR THE UNIVERSITY

E-R Diagram for the University Enterprise

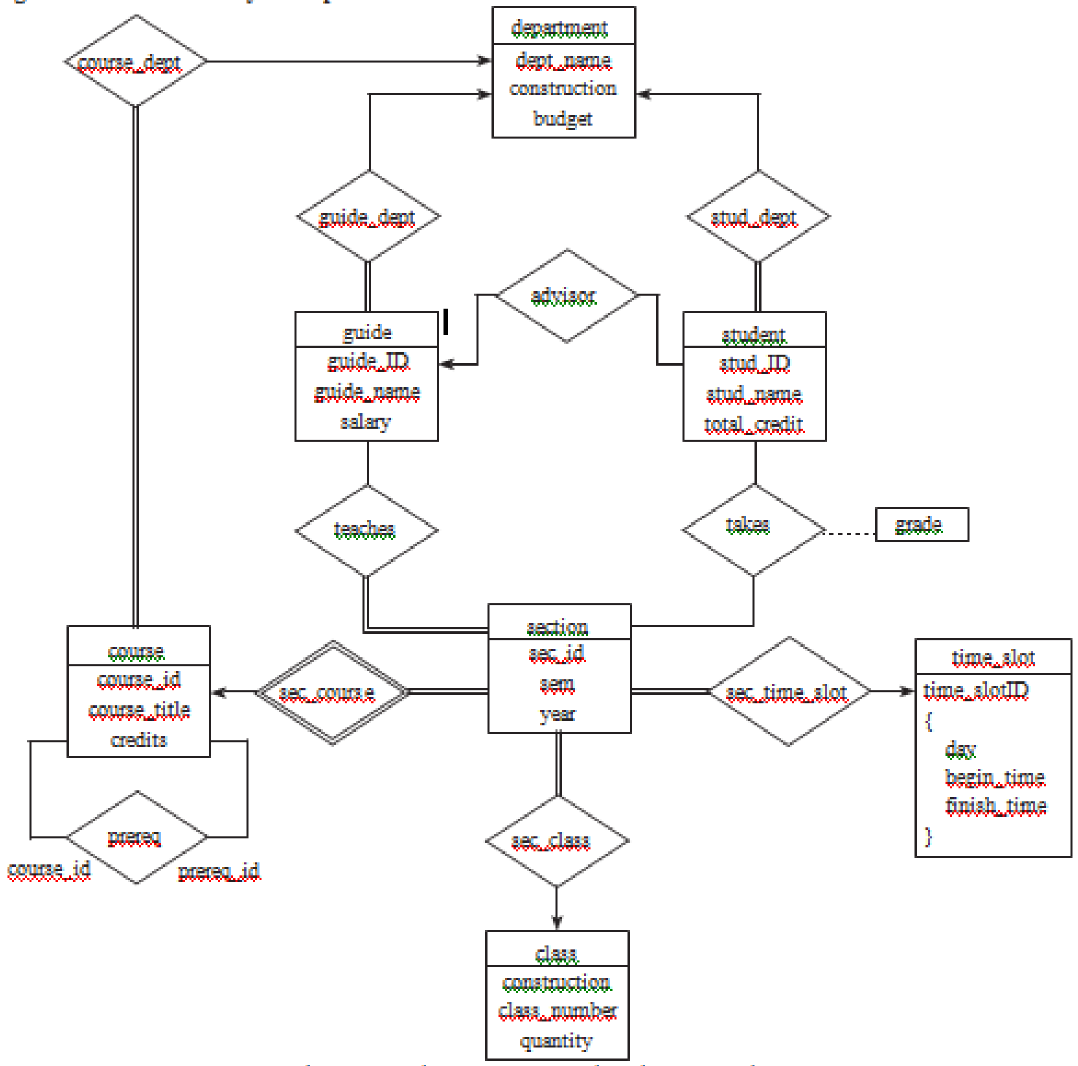


Figure: E-R Diagram for the University Enterprise

The above E-R diagram depicts the university database. It contains a constraint that every guide should have only a single related department. Besides this, it also contains a double line between guide and guide_dept. Every guide should be related with a department. There is an arrow from guide_dept to department that specifies that every guide could have one related department. Simultaneously, entity sets such as course and student contains double lines to relationship sets course_ dept and stud_dept. Also entity set 'section' to relationship set sec_time_slot. The initial two relationships return an arrow which points to the other relationship, department where the third relationship contains an arrow which points to time_slot.

The above diagram indicates the relationship set 'takes' having a descriptive attribute grade and that every student has atleast one advisor. The diagram also describes that entity set 'section' is a weak entity set, as it has the attributes such as sec_id, sem, and year. Whereas sec_course is the identifying relationship set associating weak entity set 'section' to the strong entity set course.

Reduction to Relational schemas

An E-R model can be represented in the form of relational schemas. For doing this, the entity sets and relationship sets need to be converted to relation schema for each of these sets. These schemas should be unique carrying the name same as that of entity and relationship sets.

Relation schema is similar to ER model in terms of representing real-world entities and therefore follow similar design principles.A strong entity set say with in simple descriptive attributes can be represented by a_1, a_2, a_3, a_4, ..., a_n. In a relation, every tuple in a schema differs with one entity having an entity set E.

Here, primary key of the entity set works as a primary key of the resulting schema, for schemas obtained from strong entity sets. This results from the fact that every tuple differs with a particular entity in the entity set.

For example, assume the entity set 'student' of the attributes such as stud_ID, stud_name, total_credit. Such entity set can be represented by a schema known as 'student' along with three attributes as,

student (stud_ID, stud_name,total_credit)

Observe that stud ID is the primary key of the entity set and it is also the primary key of the relation schema. Overall strong entity sets in E-R diagram except time_slot has only simple attributes. The schemas obtained from these strong entity sets are,

class (construction, class_number, quantity)

department (dept_name, contruction, budget)

course (course_id, course_title, credits)

guide (guide_id, guide_name, salary)

student (stud_ID, stud_name, total_credit)

In the above strong entity sets, both the guide and student schemas are distinct when compared to the previously used schemas which do not contain the attribute dept_name.

Entity-Relationship Design issues:

For designing an ER model, designer needs to make difficult choices like,

(i) Should the object be an entity or an attribute?

(ii) Should it be binary or ternary relationship?

(iii) Should aggregation be used or not?

(iv) Should it be an entity or relationship?

(i) Identifying if an Object is an Attribute or an Entity

Consider the student database for an institute. It is possible that each student may opt for two or more courses. The decision whether to model the course as an attribute or as an entity is difficult to take.

If a student is restricted to take only one course at a time, the course can be modeled as an attribute. However, if student opts for several courses, the course should be modeled as an entity. It can be argued that ‘the course’ can be modeled as multi valued attributes. But in real world, it is always better to model it as an entity. Since it enables the user to have the details of duration of each course, fees paid etc. Hence, when generality is more required, it is better to model an object as an entity instead of multi valued attribute.

(ii) Identifying if an Object is Binary Relationship or a Ternary Relationship

It is always desired to replace a non-binary relationship by a distinct number of binary relationship sets.

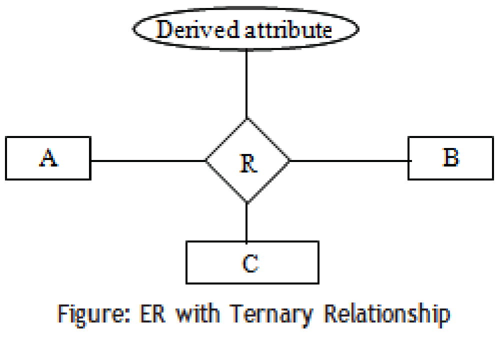


Figure: ER with Ternary Relationship

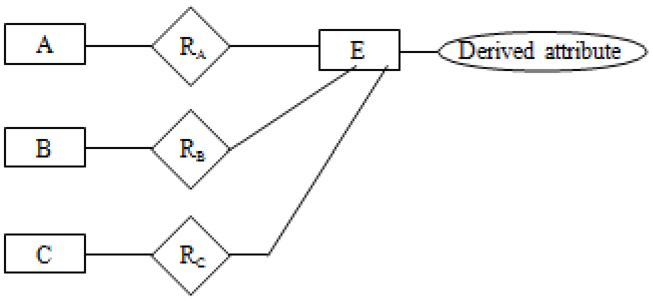


Figure: Replacing a Ternary Relation with an Entity

Consider the above ternary relationship. Now, in order to replace ternary entity with binary entities, simply create a new entity ‘E’. The attributes of the relationship set are assigned to entity ‘E’. Then generate three distinct relationships between entity ‘E’ and other three entities.

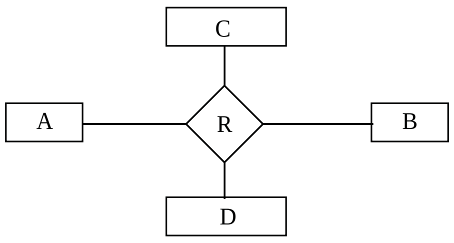
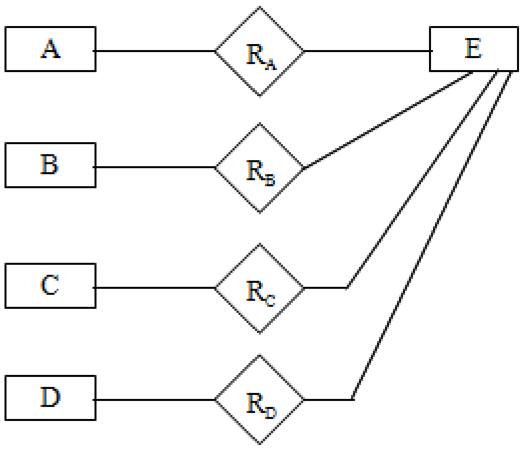


Figure: ER Diagram with Quarternery Relationship



(i) Identifying if an Object is an Aggregation or Ternary Relationship

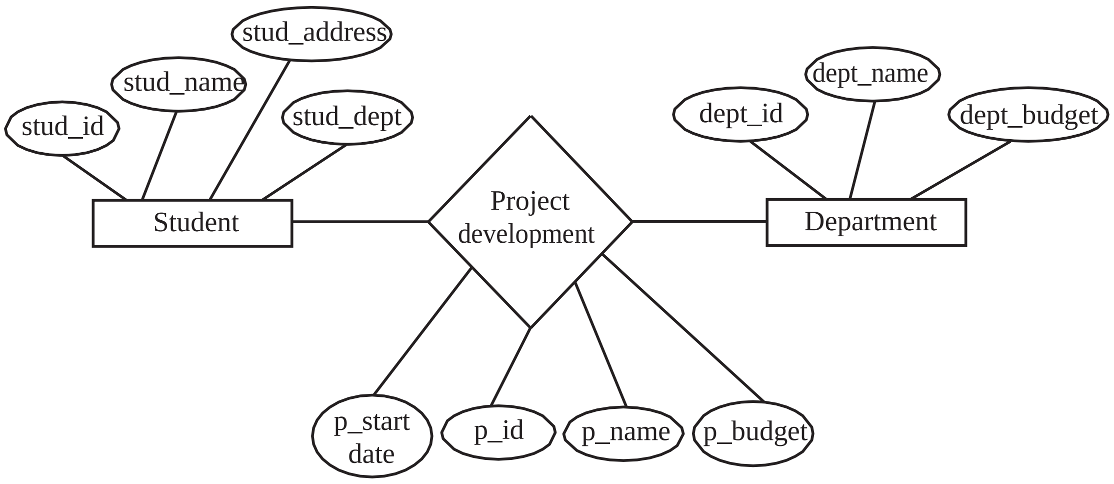
The choice of aggregation or ternary relationship depends on the following,

(i) Relationship between relationship sets and entity sets.

(ii) Integrity constraints.

(ii) Identifying if an Object is Entity or Relationship

Consider a student database for an institute. Here, it is possible that every student may develop the project based upon the budget assigned to them by the department. So,



every department has an unrestricted project budget, as shown in figure,

Figure: Entity versus Relationship

Here, the project is known to the department and the allotted budget. The method works well, if the relationship project development gets individual and unrestricted budget for every department.

<u>Relational DatabaseDesign:</u>

Features of good Relational Designs:

Schema

A schema can be defined as a complete description of database. The specifications for database schema are provided during the database design and this schema does not change frequently. Schema refinement is a process of refining the schema so as to solve the problems caused by redundantly storing the information. Redundancy in its simplest terms can be defined as duplication of data, which is main cause for all the problems which exists in database. One way to eliminate the redundant data is to make use of decomposition.

The features of good relational designs are as follows,

1. **Alternative Designing of Larger Schemas**

   The larger schemas can be designed by combining two smaller schemas by using natural join.

Example

Consider the following schema, Employee (Emp_ID, Emp_section_Id)

Department (Dept_Name, Job_title, Grade)

The above schema is considered by applying natural join on the relation as,

Emp_Dept (Emp_Id, Emp_section_Id, Dept_Name, Job_title, Grade)

Emp_Id	**Emp_Section_Id**	**Dept_Name**	**Job_Title**	**Grade**
0012	Ravi	IT	Technical writer	Level-2
0013	Nandan	CSE	Reviewer	Level-1
0013	Nandan	CSE	Reviewer	Level-1
0013	Nandan	CSE	Reviewer	Level-1
0014	Kumar	ECE	DTP operator	Level-3
0016	Teja	EEE	DTP operator	Level-3
0012	Ravi	IT	Technical writer	Level-2

Consider the above database. The tuple Emp_Id 0013 repeats the same name and same Job_Title information multiple time. This repetition wastes space as well as causes data inconsistency which inturn leads to loss of data integrity. For example, an update operation on a new record for an employee with ID 0013 has to be done 3 times i.e., it must be done for each file which stores the employees details.

In the above alternative design, it is not possible for database user to record information only for the department because, if an employee is hiring for a particular department then atleast one employee must be present in that department. Since, 'Emp_Dept' needs the values of Emp_Id, Emp_Section_Id. So, it is not possible to record information regarding newly created department unless a new employee is hired for the new department.

Alternative Designing of Smaller Schemas

The smaller schemas can be designed by using decomposition, which is the process of dividing larger schemas into smaller schemas. This division into smaller schemas is based on the functional and other dependencies which are specified by the database designers.

Example

Consider the “STUDENT” relation. STUDENT (Std_id, Name, Location)

This relation can be broken-down into two relations as follows,

(a) Location (Std_id, Location) and

(b) Name (Std_id, Name)

When a natural join operation is performed on these two schemes, the original ‘STUDENT’ relation is sustained.

STUDENT

Std_id	**Location**	**Name**
012	Hyderabad	Radha
013	Secunderabad	Meena
014	Hyderabad	Pinky
015	Secunderabad	Rani

Location		**Name**	
Std_id	**Location**	**Std_id**	**Name**
012	Hyderabad	012	Radha
013	Secunderabad	013	Meena
014	Hyderabad	014	Pinky
015	Secunderabad	015	Rani

Location ⋈ Name □ Student

Std_id	**Location**	**Name**
012	Hyderabad	Radha
013	Secunderabad	Meena
014	Hyderabad	Pinky
015	Secunderabad	Rani

No additional tuples are generated and neither data is duplicated nor data is lost. Therefore the relation STUDENT is lossless.

Location ⋈ Name □ Student 1.

Std_id	**Location**	**Name**
012	Hyderabad	Radha
013	Secunderabad	Meena
014	Hyderabad	Pinky
015	Secunderabad	Rani
012	Hyderabad	Radha
013	Secunderabad	Meena
014	Hyderabad	Pinky
015	Secunderabad	Rani

In the above table, additional tuples are generated and data is duplicated. Therefore, 'STUDENT-1' is lossy decomposition. Hence, in place of this decomposition loss-less join decomposition can be used to avoid loss of data.

Atomic Domains and First normal Form

Domain: The set of permitted values for an attribute is known as ***domain***.

Atomic Domain:

A domain is said to be atomic if all the elements of the domain can be treated as separable units (i.e., elements that cannot be divided further into smaller units). Moreover simple attributes have atomic domain and composite attributes have non-atomic domain (i.e., elements that can be further divided into smaller units).

Example

If integers are atomic then, set of integers is considered as an atomic domain. Whereas, if group of all set of integers is considered as a domain then, this set is treated as a non-atomic domain iff every individual integer is considered as ordered list of digits.

Normalization

Normalization is the process of converting a relation to a standard form by decomposing a larger relation into smaller efficient relations that depicts a good database design. Normalization technique involves a sequence of rules that are employed to test individual relations so that the database can be normalized to any degree. The main objective of normalization is to refine the design of database in order to remove data maintaining anomalies, reduce data redundancy and to eliminate data inconsistency.

The process of normalization is based on the concept of normal forms. Each and every normal form has its own set of properties and constraints. A relation is said to be in particular normal form only if it satisfies all the properties of normal form associated with that normal form. These properties are usually applied on the attributes of the relation and also on the relationship that exist between these relations.

Types of Normal Forms

The different types of normal forms based on FD's include,

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

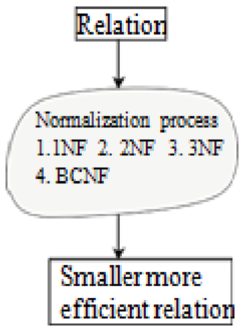


Figure: Normalization Process

1NF

A relation schema is said to be in first normal form if the attribute values in the relation are atomic i.e., there should be no repeated values in a particular column.

Converting a Relational Table to 1NF

The conversion of a relational table to 1NF can by done be performing the following steps,

Step1: In this step, the repeating groups must be removed by eliminating the null columns. This can be done by ensuring that every repeating group has an appropriate value.

Step2: In this step, a primary key among a set of attributes must be identified.

Thus, this primary key helps in uniquely identifying every attribute value.

Step3: In this step, the dependencies existing in a relation must be identified. This can be done by knowing which attribute in the relation is dependent/based on the primary key.

Example

Dept_id	**Dept_name**	**Emp _id**	**Emp_name**	**Emp_designation**	**Emp_sala ry**
2	Finance	612	Henry	Accountant	50000
		615	Morrison	Accountant Assistant	45000
		609	Peter	Finance Manager	60000
		329	Ortunga	System Analyst	46000
7	Software	311	Cena	Software Programmer	70000
		333	Rocky	Software Designer	75000

Table (i): Organization Information

The above table can be converted into 1NF by performing the following steps,

1. By eliminating the repeating groups in table (i) and assigning appropriate values to every attribute. Thus, table (i) is converted into the following table (ii) as given below,

Dept_id	**Dept_name**	**Emp_i d**	**Emp_name**	**Emp_designation**	**Emp_sala ry**
2	Finance	612	Henry	Accountant	50000
2	Finance	615	Morrison	Accountant Assistant	45000
2	Finance	609	Peter	Finance manager	60000
7	Software	329	Ortunga	System Analyst	46000
7	Software	311	Cena	Software Programmer	70000
7	Software	333	Rocky	Software Designer	75000

Table (ii): Table in 1NF

2. By identifying a primary key and uniquely determining the attribute values.

In table (ii), it is clearly observed that Dept_id cannot be considered as a primary key because it does not uniquely iden- tify the attribute values. For instance, the Dept_id = 2 identify all the three employees belonging to that Dept_id. Thus, a proper primary key must be determined so that any of the attribute values are uniquely identified. A new key consisting of Dept_id and Emp_id is created, which refers to a primary composite key. This key can be used to uniquely identify the attribute values. For instance, if Dept_id = 2 and Emp_id = 615 then the values of the attributes Dept_name, Emp_name, Emp_designation and Emp_salary are Finance, Morrison, Accountant Assistant and 45000, respectively.

3. By identifying the dependencies from the table (ii). For instance, the attributes Dept_name, Emp_name, Emp_designation and Emp_salary are dependent on the composite primary key Dept_id and Emp_id. If the Employee id is known, then its corresponding attributes such as Emp_name, Emp_designation and Emp_salary can be determined. This dependency relation can be shown as follows,

   Emp_id Emp_name, Emp_designation, Emp_salary

   In a similar way, the dept_id key can be used to determine the dept_name. This

   dependency relation can be shown as: dept_id dept_name.

Second Normal Form (2NF)

A relation is said to be in 2NF, if the following two conditions are satisfied,

(i) A relation must be in 1NF

(ii) A non-key attribute must be fully functionally dependent on primary key.

In other words, it can be said that a relation is in 2NF, if it does not contain any partial dependencies. The second condition needs to be verified only if a relation consists of a primary key that is composed of many attributes i.e., if a relation consists of a primary key with only one attribute then that table is in 2NF only if it is in 1NF.

Converting 1NF into 2NF

The conversion of a relation in 1NF (with composite primary keys) into a relation in 2NF can be done by performing the following steps,

Step1: Initially all the single key components of the relation are listed separately. Then the key component which is a composite key (i.e., the combination of all the listed keys) is specified. Each of the listed component becomes the primary key in the respec- tive table i.e., the original table is decomposed into different tables such that each table contains its related key attribute.

Step2: In this step, all of the key components identified in step 1 are assigned with their corresponding dependent attribute. These are the attributes whose value depends on the value of the key attribute.

The tables generated after performing these steps do not result in any of the data anomalies.

Example

Let us consider the dependency diagram of Employee relation obtained in 1NF.

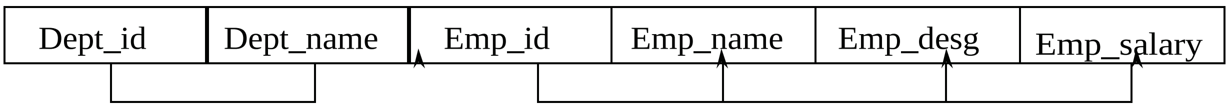


Figure: 1NF Dependency Diagram

The above dependency diagram consists of two partial dependencies,

(i) Dept_id, Dept_name.

(ii) Emp_id, Emp_name, Emp_desg, Emp_salary.

Since these partial dependencies result in data anomalies it is necessary to eliminate these dependencies. This can be done by performing the steps mentioned below,

Step 1

List the key components

(i) Dept_id

(ii) Emp_id

(iii) Dept_id, Emp_id

Here, the combination of Dept_id, Emp_id is the original composite key. Based on these key components the actual em- ployee relation is decomposed into following tables,

(i) Department

(ii) Employee

(iii) Salary.

Step 2

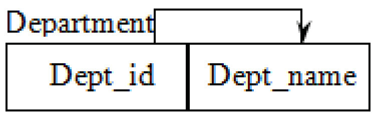
Assign the corresponding dependent attribute.

(i) Department Table

In this table, Dept_id is the primary key and Dept_name is the dependent attribute on the primary key. The relation schema and dependency diagram for Department table is,

Schema

$\text{Department(} \underbrace{\text{Dept\_id}}_{\text{Primary key}} \text{, Dept\_name)}$



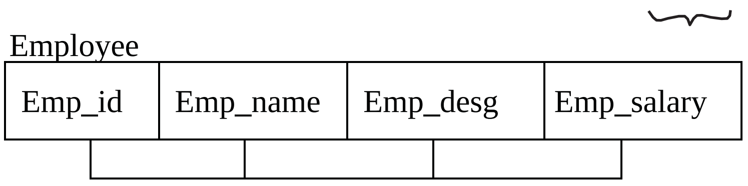
(ii) Employee Table

In this table, Emp_id is the primary key and Emp_name, Emp_desg, Emp_salary are the dependent attributes on the primary key. The relation schema and dependency diagram for Employee table is,

Schema

Employee(Emp_id, Emp_name, Emp_desg, Emp_salary)

Primary key



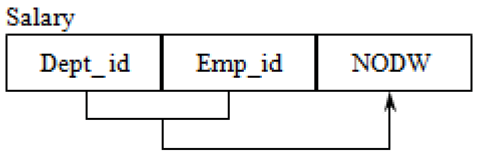
(iii) Salary Table

In this table, combination of Dept_id, Emp_id is the pri- mary key and NODW (number of days worked) is the dependent attribute on primary key.

The relation schema and dependency diagram of salary table is,

Schema

$$\text{Salary}(\underbrace{\text{Dept\_id}, \text{Emp\_id}}_{\text{Primary key}}, \text{NODW})$$



Third Normal Form (3NF)

A relation is said to be in third normal form if the fol- lowing two conditions are satisfied,

(i) A relation must be in 2NF

(ii) A relation must not have any transitive dependencies.

In other words, it can be said that a relation is in 3NF, if every determinant is a key i.e., for every functional dependency AB, A is a key. Basically, determinant is an attribute whose value determines the value of other attributes.

Converting 2NF to 3NF

The conversion of a relation in 2NF into a relation in 3NF can be done by performing the following steps,

Step1: In this step, determinants in transitive dependencies are identified and are written as a primary key for a new table.

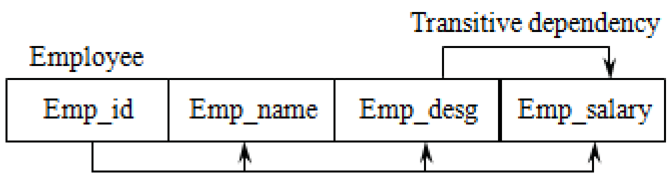
Step2: In this step, the attributes that depend on the determinant are identified along with the dependency between the attribute and its respective determinant.

Step3: In this step, the dependent attributes that were identified in step 2 are removed from every table consisting of the cor- responding transitive relationship. This deletion results in a new table that consists of determinant and also its dependent attribute that was present in the transitive relationship. The determinant acts as the primary key in the new table.

The table generated after performing these steps does not contain any inappropriate dependencies and therefore it can be said that the original table is now in 3NF.

Example

Let us consider the dependency diagram of Employee table from 2NF.



The above dependency diagram consists of a transitive dependency, Emp_desg Emp_salary, which must be eliminated in order to convert the relation into 3NF. This elimination is done by performing the above mentioned steps.

Step 1

Identify the determinant in transitive dependency Emp_desg

Step 2

Identify dependent attributes Emp_desg Emp_salary

‘Emp_salary’ is the dependent attribute since its value depends on the value of its determinant (i.e., Emp_desg)

Step 3

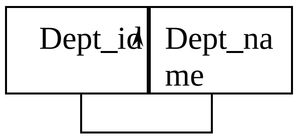
Remove the dependent attribute Emp_salary is deleted

 The dependency of “Employee” table is defined as,

Emp_id Emp_name, Emp_desg

Where Emp_desg acts as a foreign key in Employee table. The relation schema and dependency diagrams generated after completion of step 1 to step 3 for Employee relations are,

(i) **Department**



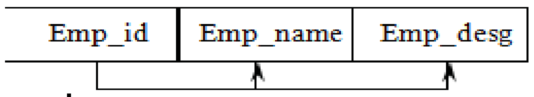
Schema

Department ($\underbrace{\text{Dept\_id}}$, Dept_name)

Primary key

Table: Dependency Diagram for Department Table

(ii) **Employee**



Schema

Employee (Emp_id, Emp_name, Emp_desg)

Primary key

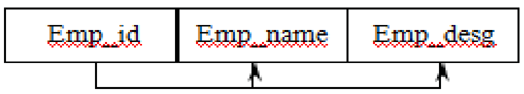
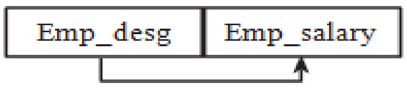


Table: Dependency Diagram for Employee Table

(iii) **Set_salary**



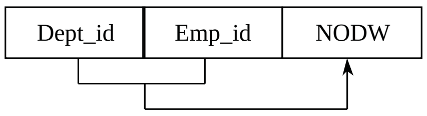
Schema

Set_salary (Emp_desg, Emp_salary)

Primary key

Table: Dependency Diagram for Salary Table

(iv) **Salary**



Schema

Salary(Dept_id, Emp_id, NODW) Primary keys

Table: Dependency Diagram for Salary

This way of converting a 2NF relation into 3NF relation helps in eliminating transitive dependency which in turn removes all the possibilities of occurrence of different data anomalies.

Boyce Codd normal form (BCNF):

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Decomposition

Decomposition is the solution to the problem caused by data redundancy. It decomposes the large schema into smaller multiple schemas. It helps in removing all the anomalies and maintains data integrity. It is possible to restrict the redundancy in Employee database by dividing it into two smaller relations/schemas as follows,

EMPLOYEE

Emp_Id	Emp_Name	Job_Section	Grade
0012	Ravi	Clerk	C
0013	Nandan	Manager	A
0013	Nandan	Manager	A
0013	Nandan	Manager	A
0014	Kumar	Secretary	B
0016	Teja	Asst. Manager	D
0012	Ravi	Clerk	C

SECTION

Emp_Section_Id	Grade
124	C
268	A
314	B
059	D

Now, it is easily possible to update Emp_Section_Id in the schema SECTION without bothering about the updations in the other tuples. To insert a new tuple, directly insert the new record in the schema SECTION (with the help of Section_Id) even if the new employee has not yet been assigned the Emp_Id. To delete the entry with the grade equal to '*A*', directly delete from SECTION schema, which does not lead to loss of other information. Thus, decomposition eliminates the problems caused by different anomalies.

Problems Related to Decomposition

The use of decomposition may lead to various problems, therefore one should be more careful with the use of decomposition. The questions that must be answered in order to use decomposition are,

(a) What problems can be caused by using decomposition?

(b) When to decompose a relation?

To answer the first question following two properties of decompositions are considered,

(a) Lossless-Join Property

This property helps to identify any instance of the original relation from the corresponding instance of the smaller relation attained after decomposition.

(b) Dependency Preservation

This property helps to enforce constraint on smaller relation in order to enforce the constraints on the original relation.

To answer the second question a number of normal forms exists. Every relation schema is in one of these normal forms and these normal forms can help to decide whether to decompose the relation schema further or not.

One of the drawbacks of decomposition is that it enforces to join the decomposed relations in order to solve the queries of the original relation. This may result in the performance degradation if such queries are common. In order to improve the performance it is necessary to ignore the problems caused by redundancy and decomposition of the relation.

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

X → Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

TYPES OF FUNCTIONAL DEPENDENCIES:

An entity set can have the following functionaldependencies,

1. Full functional dependency
2. Partial functional dependency
3. Transient or transitive functional dependency.

1. Full Functional Dependency

If *x* and *y* are attributes of an entity set in a table such that *y* is functionally dependent only on *x*, but not on any proper subset of *x*, then this type of dependency is called full functional dependency.

Example

Suppose the roll number and subject name specify the marks of a student in particular subject. In this case, it is not mandatory that a student will get good marks in all the subjects.

Therefore, marks depend on the subject name.

i.e., RollNumber, SubjectName → Marks

2. Partial Functional Dependency

If *x* and *y* are attributes of an entity set in a table and *y* is functionally dependent on *x* such that elimination of some attributes from *x* does not effect the dependency, then this type of dependency is called partial functional dependency. That is, *y* is partially dependent on *x*.

Example

During examinations, any subject can be randomly held in any examination hall. Therefore, hall number depends on subject name as well as roll numbers. In this case, hall number's dependency on subject name is partial because it also depends on roll number as well.

SubjectName → HallNumber

3. Transient or Transitive Functional Dependency

If *x*, *y*, *z* are attributes of an entity set in a table such that *x* is functionally dependent on *y* and *y* is functionally dependent on *z*, then *z* will be transitively dependent on *x* through *y*.

Example

For examinations to be held, if students are dependent on teachers and teachers are dependent on management. Then management will be transitively dependent on students through teachers.

Students → Teachers

Teachers → Management

4. Multi valued Functional Dependency

In Multi valued functional dependency, entities of the dependent set are not dependent on each other. i.e. If a → {b, c} and there exists no functional dependency between b and c, then it is called a multi valued functional dependency.

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18
45	abc	19

Here, roll_no → {name, age} is a multivalued functional dependency, since the dependents name & age are not dependent on each other (i.e. name → age or age → name doesn't exist !)

Algorithms for Decomposition

Decomposition to BCNF

Before applying the BCNF decomposition algorithm to the given relation, it is necessary to test if the relation is in Boyce-Codd Normal Form. After the test, if it is found that the given relation is not in BCNF, we can decompose it further to create relations in BCNF.

There are following cases which require to be tested if the given relation schema R satisfies the BCNF rule:

Case 1: Check and test, if a nontrivial dependency **α -> β** violate the BCNF rule, evaluate and compute α^+ , i.e., the attribute closure of α. Also, verify that α+ includes all the attributes of the given relation R. It means it should be the superkey of relation R.

Case 2: If the given relation R is in BCNF, it is not required to test all the dependencies in F^+. It only requires determining and checking the dependencies in the provided dependency set F for the BCNF test. It is because if no dependency in F causes a violation of BCNF, consequently, none of the F^+ dependency will cause any violation of BCNF.

BCNF Decomposition Algorithm

This algorithm is used if the given relation R is decomposed in several relations R_1, R_2,…, R_n because it was not present in the BCNF. Thus,

For every subset α of attributes in the relation R_i, we need to check that α^+ (an attribute closure of α under F) either includes all the attributes of the relation R_i or no attribute of R_i-α.

```
result={R};
done=false;
compute $F^+$;
while (not done) do
        if (there is a schema Ri in result that is not in BCNF)
                then begin
                        let $\alpha$->$\beta$ be a nontrivial functional dependency that holds
                        on $R_i$ such that $\alpha$->$R_i$ is not in $F^+$, and $\alpha \cap \beta = \varnothing$;
                        result=(result-$R_i$) U ($R_i$-$\beta$) U ($\alpha$,$\beta$);
                end
                else done=true;
```

This algorithm is used for decomposing the given relation R into its several decomposers. This algorithm uses dependencies that show the violation of BCNF for performing the decomposition of the relation R. Thus, such an algorithm not only generates the decomposers of relation R in BCNF but is also a lossless decomposition. It means there occurs no data loss while decomposing the given relation R into R_1, R_2, and so on…

The BCNF decomposition algorithm takes time exponential in the size of the initial relation schema R. With this, a drawback of this algorithm is that it may unnecessarily decompose the given relation R, i.e., over-normalizing the relation. Although decomposing algorithms for BCNF and 4NF are similar, except for a difference. The fourth normal form works on **multivalued dependencies,** whereas BCNF focuses on the **functional dependencies**. The multivalued dependencies help to reduce some form of repetition of the data, which is not understandable in terms of functional dependencies.

Decomposition to 3NF

The decomposition algorithm for 3NF ensures the preservation of dependencies by explicitly building a schema for each dependency in the canonical cover. It guarantees that at least one schema must hold a candidate key for the one being decomposed, which in turn ensures the decomposition generated to be a lossless decomposition.

3NF Decomposition Algorithm

```
let Fc be a canonical cover for F;
i=0;
for each functional dependency $\alpha \rightarrow \beta$ in $F_c$
        i = i+1;
R = $\alpha\beta$;
If none of the schemas $R_j$, j=1,2,…I holds a candidate key for R
Then
        i = i+1;
        $R_i$= any candidate key for R;
/* Optionally, remove the repetitive relations*/
Repeat
        If any schema $R_j$ is contained in another schema Rk
 Then
/* Delete $R_j$ */
$R_j = R_i$;
i = i-1;
until no more Rjs can be deleted
return ($R_1$, $R_2$, . . . ,$R_i$)
```

Here, R is the given relation, and F is the given set of functional dependency for which F_c maintains the canonical cover. R_1, R_2, . . . , R_i are the decomposed parts of the given relation R. Thus, this algorithm preserves the dependency as well as generates the lossless decomposition of relation R.

A 3NF algorithm is also known as a **3NF synthesis algorithm.** It is called so because the normal form works on a dependency set, and instead of repeatedly decomposing the initial schema, it adds one schema at a time.

Drawbacks of 3NF Decomposing Algorithm

- The result of the decomposing algorithm is not uniquely defined because a set of functional dependencies can hold more than one canonical cover.
- In some cases, the result of the algorithm depends on the order in which it considers the dependencies in F_c.
- If the given relation is already present in the third normal form, then also it may decompose a relation.

MUILTIVALUED DEPENDENCY:

- Multi valued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multi valued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces two colours (white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multi valued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

1. BIKE_MODEL → → MANUF_YEAR

2. BIKE_MODEL → → COLOR

   This can be read as "BIKE_MODEL multi determined MANUF_YEAR" and "BIKE_MODEL multi determined COLOR".

MORE NORMAL FORMS:

4 NF(fourth normal form):

A relation R is in 4NF if it is in BCNF and there is no non-trivial multiv alued dependency.

For a dependency A->B, if for a single value of A, multiple values of B exist, then the relation will be a multi-valued dependency.

Example

Consider the following table

Regno	Phoneno	Qualification
1	P1	Diploma
1	P1	B.Tech
1	P1	M.Tech
1	P2	Diploma
1	P2	B.Tech
1	P2	M.Tech

Here,

regno->-> phoneno

regno->-> qualification.

Both are non trivial MVD

The given relation is in BCNF [since no functional dependency exists]. But the above table is not in 4NF [since there is a non trivial MVD].

Anomalies

It also suffers with anomalies which are as follows −

- Insertion anomaly: If we want to insert a new phoneno for regno3 then we have to insert 3 rows, because for each phoneno we have stored all three combinations of qualification.
- Deletion anomaly: If we want to delete the qualification diploma, then we have to delete it in more than one place.
- Updation anomaly: If we want to update the qualification diploma to IT, then we have to update in more than one place.

<u>4NF decomposition:</u>

If R(XYZP) has X->->Y and X->->Z then, R is decomposed to R1(XY) and R2(XZP).

=> R(regno, phoneno, qualification) is decomposed to R1(regno, phoneno) and R2(regno, qualification).

R1

Regno	**Phoneno**
1	P1
1	P2

R2

Regno	**Qualification**
1	Diploma
1	B.Tech
1	M.Tech

All the anomalies discussed above are removed. The above two relations are in 4NF.

Now, regno->->phoneno is trivial MVD (since {regno} U {phoneno}=R1)=>R1 is in 4NF.

Regno->-> qualification is trivial MVD (since {regno} U {qualification} =R2)=> R2 is in 4NF.

<u>5NF (FIFTH NORMAL FORM:</u>

The 5NF (Fifth Normal Form) is also known as project-join normal form. A relation is in Fifth Normal Form (5NF), if it is in 4NF, and won't have lossless decomposition into smaller tables.

Example

The below relation violates the Fifth Normal Form (5NF) of Normalization −

<Employee>

EmpName	**EmpSkills**	**EmpJob** (Assigned Work)
David	Java	E145
John	JavaScript	E146
Jamie	jQuery	E146
Emma	Java	E147

The above relation can be decomposed into the following three tables; therefore, it is not in 5NF −**<EmployeeSkills>**

EmpName	**EmpSkills**
David	Java
John	JavaScript
Jamie	jQuery

Emma	Java

The following is the <EmployeeJob> relation that displays the jobs assigned to each employee –

<EmployeeJob>

EmpName	**EmpJob**
David	E145
John	E146
Jamie	E146
Emma	E147

Here is the skills that are related to the assigned jobs –

<JobSkills>

EmpSkills	**EmpJob**
Java	E145
JavaScript	E146
jQuery	E146
Java	E147

Join Dependency –

{(EmpName, EmpSkills), (EmpName, EmpJob), (EmpSkills, EmpJob)}

The above relations have join dependency, so they are not in 5NF. That would mean that a join relation of the above three relations is equal to our original relation **<Employee>**.