Because the relational model has become dominant in modern database design, thus much of the attention is being given to the layers closest to the conceptual level. Therefore, E-R model has become the point of consideration for this reason; it's a tool which is commonly used to

- Translate different views of data among managers, users, and programmers to fit into a common framework.
- Define data processing and constraint requirements to help us meet the different views.
- Help implement the database.

The E-R model has been used for several years and is now widely accepted. The E-R approach to conceptual database modeling was first described in 1976 by Peter Chen in a landmark paper entitled, "The Entity-Relationship Model: Toward a Unified View of Data.". Chen's article generated much additional research effort, which yielded the refinements found in the current E-R model.

E-R MODEL COMPONENTS

The E-R model forms the basis of E-R diagrams, which represent the conceptual data- base as viewed by the end user. These diagrams depict the E-R model's three main components: *attributes, entities(also called Entity types),* and *relationships(also called Relationship types).* Because an entity represents a real- world *object,* the words *entity* and *object* are often used interchangeably.

Entities

An entity at the E-R modeling level actually refers to the *entity set* and *not* to a single entity occurrence. *In other words, the word "entity" in the E-R model corresponds to a table and not to a row in the relational environment.* This is the reason that the Entity is also called Entity type.

Entity Type	An object or concept that is identified by the enterprise as having an independent existence.

The basic concept of the ER model is an entity type, which represents a set of 'objects' in the 'real world' with the same properties. An entity type has an independent existence and can be an object with a physical (or 'real') existence or an object with a conceptual (or 'abstract') existence, as listed in Figure Lecture5_5.2. Note that we are only able to give a working definition of an entity type as no strict formal definition exists. This means that different designers may identify different entities. An entity (type) is represented by a rectangle containing the entity's name.

Physical existence	
Staff	Part
Property	Supplier
Customer	Product
Conceptual existence	
Viewing	Sale
Inspection	Work experience

Figure Lecture5_5.2

Entity:	An instance of an entity type that is uniquely identifiable.

The E-R model refers to a specific table row as an entity instance or entity occurrence. Each uniquely identifiable instance of an entity type is referred to simply as an entity. Other authors may refer to our definition of an entity as an entity occurrence or entity instance.

We identify each entity type by a name and a list of properties. A database normally contains many different entity types. Examples of entity types are such as Staff, Branch, and Student. Although an entity type has a distinct set of attributes, each entity has its own values for each attribute.

Attributes

Attributes are represented by ovals and are connected to the entity with a line. Each oval contains the name of the attribute it represents. For example, the STUDENT entity includes the attributes STU_LNAME, STU_FNAME, and STU_INITIAL, as shown in Figure Lecture5_4.6.

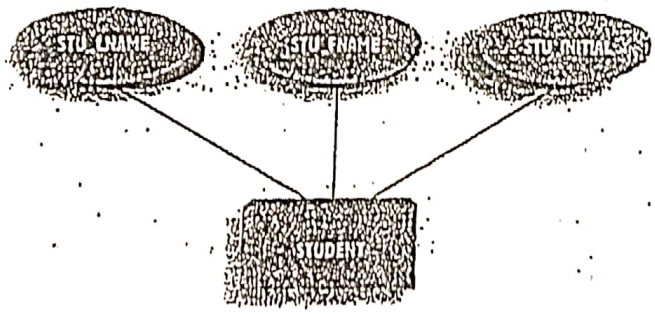


Figure Lecture5_4.6

Attribute	A property of an entity or a relationship type.

The particular properties of entities are called attributes. For example, a Branch entity may be described by the branch number (Branch_No), address (Address), phone number (Tel_No), und fax number (Fax_No). The attributes of an entity hold values that describe each entity. The values held by attributes represent the main part of the data stored in the database. A relationship that associates entities can also have attributes similar to those of an entity type.

Attributes also have a domain. A domain is the attribute's set of possible values. Each attribute is associated with a set of values called a domain. The domain defines the potential values that an attribute may hold. For example, the domain for the (numeric) attribute Grade Point Average (GPA) is written (0,4) because the lowest possible GPA value is 0 and the largest possible value is 4.The domain for the (character) attribute SEX consists of only two possibilities, M or F (or some other equivalent code). The domain for a company's date of hire consists of all dates that fit into a range (for example, company startup date to current date).

Attribute domain	A set of values that may be assigned to an attribute.

Attributes may share a domain. For instance, a student address and a professor address share the same domain of all possible addresses. In fact, the data dictionary may let a newly declared attribute inherit the characteristics of an existing attribute, if the same attribute name is used. For example, the PROFESSOR and STUDENT entities may each have an attribute named ADDRESS.

The domain of the first name (FName) attribute is more difficult to define, as it consists of all first names. It is certainly a character string, but it might consist not only of letters but also of hyphens or other special characters. A fully developed data model includes the domains of each attribute in the ER model.

Primary keys (key attributes) are underlined in the E-R diagram shown in Figure 4.7. Key attributes are also underlined in a frequently used table structure shorthand using the format

TABLE NAME (KEY ATTRIBUT~1,ATTRIBUTE2,ATTRIBUTE3 ..... ATTRIBUTEK)

For example, Figure 4.7'5 E-R model may be represented by:

CAR (CAR_ID_NUM, MOD_CODE, CAR_YEAR, CAR_COLOR)

Given the naming conventions, it is easy to see that the attribute named MOD_CODE is likely to be a foreign key that is used to connect the CAR table to some other table. In this case, the other table probably contains model- specific attributes, storing descriptive entries such as "2-door sports coupe" or "4-door sedan" or "station wagon."

Prepared and Compiled by
S. M. KHALID JAMAL

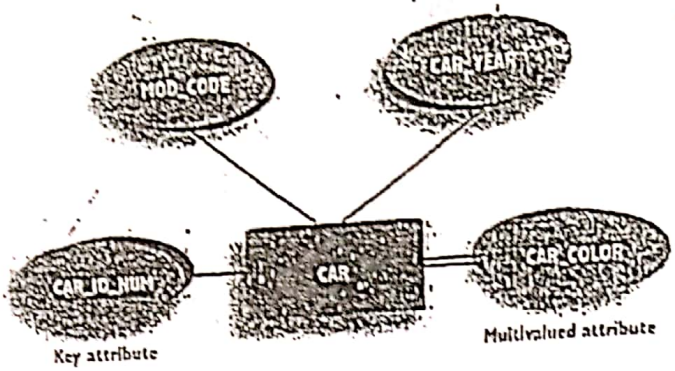


Figure Lecture5_4.7

Ideally a primary key is composed of only a single attribute. For example, Database Table Lecture5_4.1 shows a single-attribute primary key named CLASS_CODE. However, it is possible to have composite keys that are primary keys composed of more than one attribute. For instance, a University's Database Administrator may decide to identify each CLASS entity instance (occurrence) by using a composite primary key composed of the combination of CRS_CODE and CLASS_SECTION, instead of using CLASS_CODE. Either approach uniquely identifies each entity instance. Given Database Table Lecture5_4.1's current structure, CLASS_CODE is the primary key and the combination of CRS_CODE and CLASS_SECTION is a proper candidate key: If the CLASS_CODE attribute is deleted from the CLASS entity the candidate key (CRS_CODE and CLASS_SECTION) becomes an acceptable composite primary key.

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
10010	ACCT-211	1	MWF 8:00-8:50 a.m.	BUS311	105
10013	ACCT-211	2	MWF 9:00-9:50 a.m.	BUS200	105
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS200	142
10015	ACCT-212	1	MWF 10:00-10:50 a.m.	BUS311	311
10016	ACCT-212	2	TTh 1:00-2:40 p.m.	BUS200	311
10017	CIS-220	1	MWF 9:00-9:50 a.m.	KLR209	220
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10019	CIS-220	3	MWF 10:00-10:50 a.m.	KLR209	228
10020	CIS-420	1	W 6:00-8:40 p.m.	KLR209	162
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR209	114
10022	QM-261	2	TTh 1:00-2:15 p.m.	KLR210	114
10023	QM-362	1	MWF 11:00-11:50 a.m.	KLR210	162
10024	QM-362	2	TTh 2:30-3:45 p.m.	KLR209	162

Database Table Lecture5_4.1

If the Database Table Lecture5_4.1's class code is used as the primary key, the CLASS entity may be represented in shorthand form by

CLASS (CLASS_CODE, CRS_CODE, CLASS SECTION, CLASS_TIME, CLASS_ROOM, PROF_NUM)

On the other hand if CLASS_CODE is deleted and the composite primary key is the combination of CRS_CODE and CLASS_SECTION, the CLASS entity may be represented by

CLASS (CRS_CODE, CLASS_SECTION, CLASS_TIME, CLASS_ROOM, PROF_NUM)

Attributes are classified as simple or composite. A simple attribute cannot be subdivided. Sometimes they are also called atomic attributes because of the property that they cannot be further subdivided For example, age, sex, and marital Status would be classified as simple attributes.

Simple attribute	An attribute composed of a single component with an independent existence.

A composite attribute, not to be confused with a composite key, is an attribute that can be further subdivided to yield additional attributes. For example, the attribute ADDRESS can be subdivided into street, city, State, and ZIP code. Similarly: the attribute PHONE_NUMBER can be subdivided into area code and exchange number: To facilitate detailed queries, it is often appropriate to change composite attributes into a series of simple attributes.

Composite attribute	An attribute composed of multiple components, each with an independent existence.

The decision to model the Address attribute as a simple attribute or to subdivide the attribute into Street, Area, City, and Postcode is dependent on whether the user view of the-model refers to the Address attribute as a single unit or as individual components.

The majority of attributes are single-valued for a particular entity. For example, the Branch entity has a single-value for the branch number (Branch_No) attribute (for example' n3), and t11erefore the Branch_No attribute is referred to as being single-valued.

Single-valued attribute	An attribute that holds a single value for a single entity.

A **single-valued attribute** can have only a single value. For example, a person can have only one social security number; and a manufactured part can have only one serial number: Keep in mind that a single-valued attribute is not necessarily a simple attribute. For instance, the part's serial number SE-08-02-189935 is single-valued, but it is a composite attribute because it can be subdivided into the region in which the part was produced, SE, the plant within that region, 08, the shift within the plant, 02, and the part number; 189935.

Some attributes have multiple values for a particular entity. For example, the Branch entity may have multiple values for the branch telephone number (Tel_No) attribute (for example, 0171-886-1212 and 0171-886-1233) and therefore the Tel_No attribute in this case would be multi-valued. A multi-valued attribute may have a set of numbers with upper and lower limits. For example, the Tel_No attribute of a branch may have between one and ten values. In other words, a branch may have a minimum of a single telephone number or a maximum of ten telephones.

Multi-valued attribute	An attribute that holds multiple values for a single entity.

Multi-valued attributes can have many values. For instance, a person may have several college degrees. Similarly, a car's color may be subdivided into colors of roof, body, and trim. In an E-R model the multi-valued attributes are shown by a double line connecting the attribute to the entity. The E-R diagram in Figure Lecture5_4.7 contains all the components we have introduced thus far: As you examine Figure Lecture5_4.7, note that the CAR_ID_NUM is the primary key and CAR_COLOR is a Multivalued attribute of the CAR entity. (We will assume that each car is identified by a unique registration number or CAR_ID_NUM.

Although the conceptual model can handle M:N relationships and mute-valued attributes, you should not implement them in the relational DBMS. So if multi-valued attributes exist, the designer must decide on one of two possible courses of action:

i) .Within the original entity, create several new attributes, one for each of the original multi-valued attribute's components. For example, we can split the CAR entity's attribute CAR~COLOR to create the new attributes, CAR- TOPCOLOR, CAR_BODYCOLOR, and CAR- TRIMCOLOR shown in Figure Lecture5_4.8. and assign them to the CAR entity.

ii) .Create a new entity composed of the original multi-valued attribute's components. (See Figure Lecture5_4.9.) The new (independent) COLOR entity is then related to the original CAR entity in a I:M relationship. Note that such a change allows the designer to define color for different sections of the car

Section	Color
Top	White
Body	Blue
Trim	Green
Interior	Blue

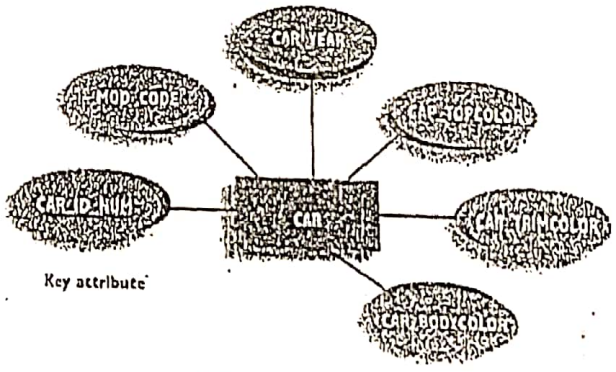


Figure Lecture5_4.8

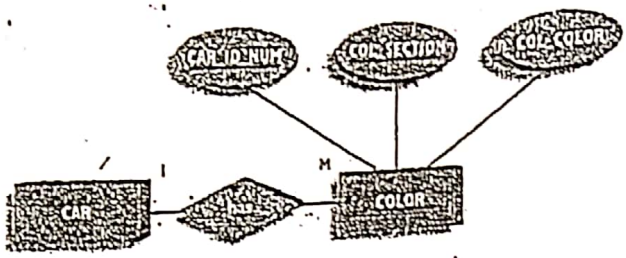


Figure Lecture5_4.9

Finally, an attribute may be classified as a derived attribute. A derived attribute need not be physically stored within the database; instead, it is derived by using an algorithm. For example, a person's AGE attribute can be derived by subtracting the date of birth (DOB) from the current date. Similarly, the total cost of an order can be derived by multiplying the quantity ordered by the unit price. Or the estimated average speed can be derived by dividing trip distance by the time spent en route. A derived attribute is indicated by a dotted line connecting the attribute and the entity. (See Figure Lecture5_4.10.) The decision to use derived attributes depends on processing requirements and the constraints placed on a particular application. The designer should be able to balance the design in accordance with such constraints.

Derived attribute	An attribute that represents a value that is derivable from the value of a related attribute or set of attributes, not necessarily in the same entity.

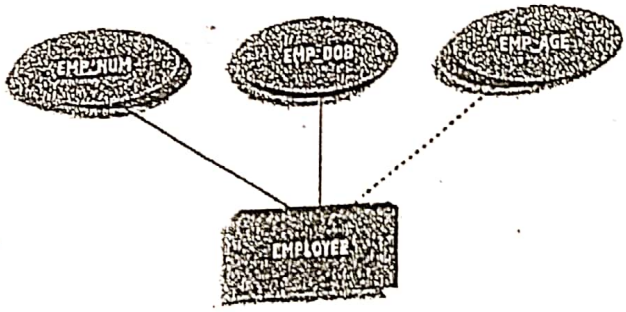


Figure Lecture5_4.10

In some cases, the value of an attribute is derived from the entities in the same entity type. For example, the total number of staff (Total_Staff) attribute of the Staff entity type can be calculated by counting the total number of Staff entities

Prepared and Compiled by:
S. M. KHALID JAMAL

Derived attributes may also involve the association of attributes of different entities. For example, consider an attribute called Deposit of a Rental_Agreement entity. The value of the rental deposit (Deposit) attribute associated with a rental agreement is calculated as twice the monthly rent for the property. Therefore, the value of the Depict attribute of the Rental_Agreement entity is derived from the Rent attribute of the Property_for_Rent entity.

A relationship is an association between entities. Each relationship is identified so that its name is descriptive of the relationship. Ideally, the relationship name is an active verb, but passive verbs are acceptable, too. Verbs such as takes, teaches, and employs make good relationship names. For example, a student takes a class, a professor teaches a class, a department employs a professor; and so on.

Relationship type	A meaningful association among entity types.

A relationship type is a set of associations between two (or more) participating entity types. Each relationship type is given a name that describes its function. For example, the Owner entity is associated with the Property_for_Rent entity through the relationship called Owns. As with entities, it is necessary to distinguish between the terms 'relationship type' and 'relationship'.

Figure Lecture5_4.11

Relationships are represented by diamond-shaped symbols. Figure Lecture5_4.11 depicts a relationship between two entities (also known as the participants in the relationship) named PROFESSOR and CLASS, respectively. Lines are used to connect the entities to one another; passing through the diamond-shaped relationship symbol.

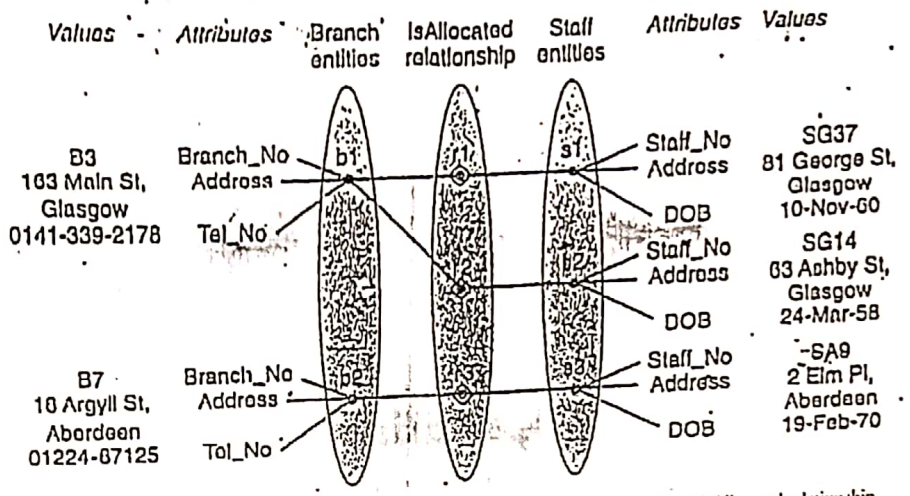


Figure Lecture5_5.5 A Semantic Net Model illustrating individual occurrences of the IsAllocated relationship.

Relationship	An association of entities where the association includes one entity from each participating entity type.

Each uniquely identifiable occurrence of a relationship type is referred to simply as a relationship. A relationship indicates the particular entities that are related. Other authors may refer to our definition of a relationship as a relationship occurrence or relationship instance. Throughout this chapter, we use only the terms 'relationship type' or 'relationship'. As with the term entity, we use the more general term 'relationship' when the meaning is obvious.

The relationship IsAllocated indicates an association between Branch and Staff entities, where each occurrence of the IsAllocated relationship associates one Branch entity with one Staff entity. Figure 5.5 represents individual occurrences of the IsAllocated relationship using a diagram called a semantic net. The semantic net is an object-level diagram in which the ○ symbol represents entities and the ◇ symbol represents relationships.

To simplify the semantic net diagram, only some of the attributes of the Branch and Staff entities are represented in Figure 5.5. The Branch entity type is reduced to three attributes: Branch_No, Address, and Tel_No and the Staff entity type has three attributes: Staff_No, Address, and DOB (Date of Birth). Each attribute holds a value tie from its associated domain. For example, the value for the Branch_No attribute for Branch entity (b1) is B3.

There are three relationships (r1, r2, and r3) that describe the association of the Branch entity with. the Staff entity. The relationships are shown by lines, which join each participating Branch entity with the associated Staff entity. For example, relationship r1 represents the association between Brunch entity b1 and Staff entity s1.

Prepared and Compiled by
S. M. KHALID JAMAL

If we represented an enterprise using semantic nets, it would be difficult to understand due to the level of detail. We can more easily represent the relationships between entities in an enterprise using the concepts of the Entity-Relationship model.

To reduce the level of detail shown in a "single ER diagram, often only the attributes that represent the primary key of each entity are displayed and in some cases, no attributes are shown at all.

Degree of a relationship	The number of participating entities in-a relationship.

The entities involved in a particular relationship are referred to as participants in that relationship. The number of participants in a relationship is called the degree of that relationship. Therefore, the degree of a relationship indicates the number of entities involved in a relationship. A relationship of degree two is called, binary. An example of a binary relationship is Owns, with two participating entities namely Owner and Property_for_Rent. Figure Lecture5_5.7(a) diagrammatically represents binary relationship

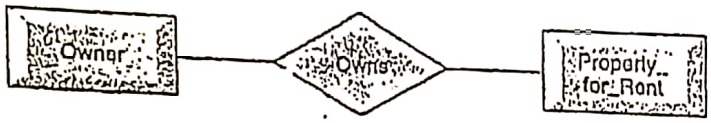


Figure Lecture5_5.7(a)

A relationship of degree three is called ternary. An example of a ternary relationship is *SetsUp* with three participating entities, namely Client, Staff, and Interview, The purpose of this relationship is to represent the situation where a member of staff is responsible for setting up an interview with a client, Figure Lecture5_5.7(b) diagrammatically represents the ternary relationship *SetsUp*.

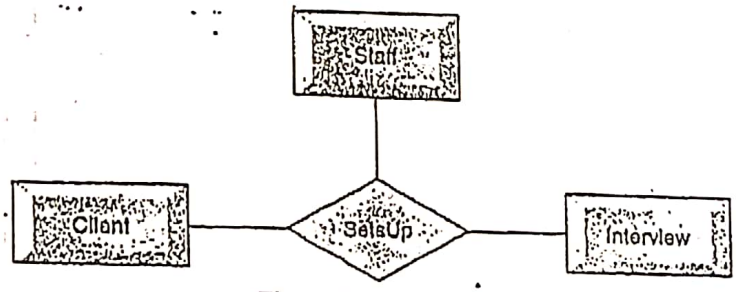


Figure Lecture5_5.7(b)

A relationship of degree four is called quaternary. An example of a quaternary relationship is *Arranges* with four participating entities, namely Buyer, Solicitor, Financial_Institution, and Bid. This relationship represents the situation where a buyer, advised by a solicitor, and supported by a financial institution, places a bill for a property. Figure Lecture5_5.7(c) diagrammatically represents the quaternary relationship *Arranges*.

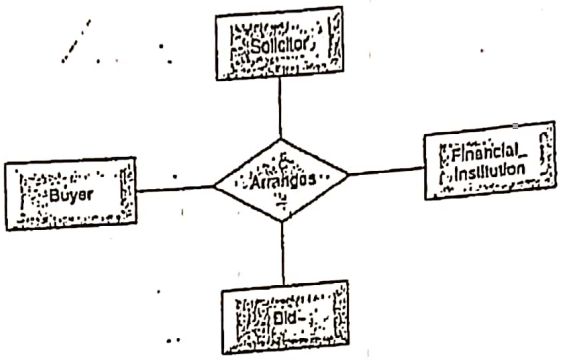


Figure Lecture5_5.7(c)

A unary relationship exists when an association is maintained within a single entity. This association exists in a recursive relationship.

Recursive relationship	A relationship where the *same* entity participates more than once in different role

Consider a recursive relationship called Supervises, which represents an association of staff with a supervisor where the supervisor is also a member of staff. In other words, the Staff entity participates twice in the Supervises relationship; the first participation as a supervisor, and the second participation as a member of staff who is supervised (supervisee). Recursive relationships are sometimes called unary relationships.

Relationships may be given role names to indicate the purpose that each participating entity plays in a relationship. Role names are important for recursive relationships to determine tile function of each participation. The use of role names to describe the Supervises recursive relationship is shown in figure Lecture5_5.8. The first participation of the Staff entity in the supervise relationship is given the role name supervisor and the second participation is given the role name Supervisee.

Role names may also be used when two entities are associated through more than one relationship. For example, the Staff and Branch entities are associated through two distinct relationships called Manages and IsAllocated. As shown in figure Lecture5_5.9, the use of role names clarifies the purpose of each relationship. For example, in the case of the Staff Manages Branch, a member of staff (Staff entity) given, the role name 'Manager' manages a branch (Branch entity) given the role name 'Branch Office'.

Similarly, for Branch IsAllocated Staff, a branch, given the role name 'Branch Office' is allocated staff, given the role name 'Member of Staff'.

Role names are usually not required if the function of the participating entities in a relationship is unambiguous.

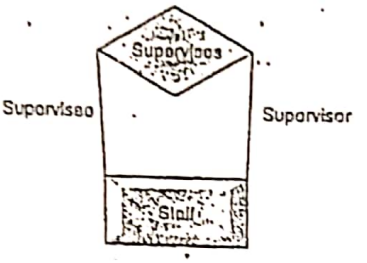


Figure Lecture5_5.8

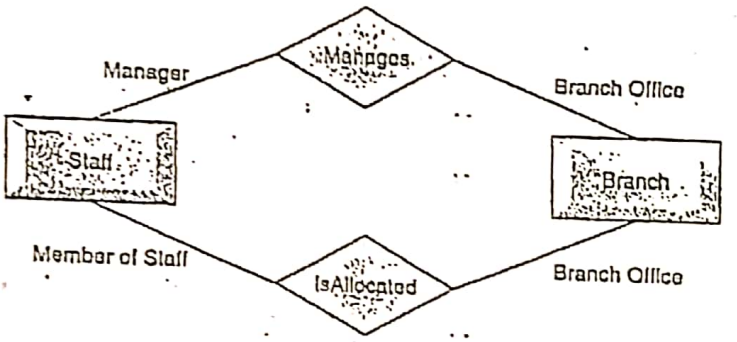


Figure Lecture5_5.9

Binary relationships are most common. In fact, to simplify the conceptual design, most higher-order (ternary and higher) relationships are decomposed into appropriate equivalent binary relationships whenever possible.

Although most relationships are binary, the use of ternary and higher-order relation- ships does allow the designer some latitude regarding the semantics of a problem. For example, note the relationships (and their consequences!) in Figure 4.12:

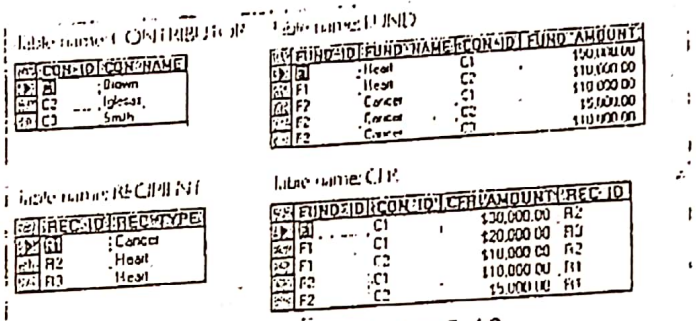
1. People or institutions in the CONTRIBUTOR group donate money to a special research FUND. (The research funds may be placed into several

categories. For example, if the fund is designed to support medical research, the categories may include cancer, AIDS, heart disease, and so on.)

2. The researchers who are found in the RECIPIENT entity are funded through the FUND contents.

3. The relationships are all classified as M:N. For example, contributors can make donations to many funds. A fund can have many donors. A fund may support many researchers who become the FUND recipients, and a researcher may draw support from many funds.

4. The three entities CONTRIBUTOR. RECIPIENT; and FUND are related through the ternary relationship we have labeled CFR. Because a single verb cannot express a ternary relationship, we elected to use the acronym CFR to indicate its role in defining the relationships between the CONTRIBUTOR, the FUND, and the RECIPIENT. Thus CFR enables the database end user to identify the fund's contributors for each recipient, as shown in Database Table Lecture5_4.2.



Database Table Lecture5_4.2

As you examine the table contents in Database Table Lecture5_4.2, note that it is possible to track all transactions. For instance, researcher R I. specializing in cancer research, has received a total of $15,000, divided as follows: $10,000 from contributor CI (Brown) and $5,000 from contributor C2 (Iglesas). Researcher R2, specializing in heart research, received $40,000--$30,000 of which came from contributor C I and $1 0,000 of which came from contributor C2.In addition, the CFR (ternary) relationship allows us to relate the available funds to the contributors. For example, the table contents indicate that contributor C I donated $50,000 to heart research-$30,000 of which was received by researcher R2, while the remaining $20,000 was received by researcher R3.

The ternary relationship described here has an effect on the type and extent of the data available to a database query: For example, if 500 people make donations to the heart portion of the fund and we do not use a ternary relationship, there is no way to identify

the specific source of those funds when the recipient receives a heart grant Therefore, if it is necessary to keep track of specific contributors to the fund and the recipients of those funds, the ternary relationship is necessary; As the preceding example demonstrates, you should keep in mind that a ternary relationship is not always equivalent to several I:M relationships. Consequently the designer must consider the semantics of the problem that is being addressed; desirable simplifications may not match end user needs!

The attributes can also be assigned to relationships For example, consider the relationship Views, which associates the Client and Property_for_Rent entities. We may wish to record the date the property was viewed by the client and any comments made by the client regarding the suitability or other wise of the property. This information is associated with the Views relationship rather than the Client or the Property_for_Rent entities. As shown in Figure Lecture5_5.10 we create attributes called Date_View and Comments to store this information ant assign them to the Views relationship.

The presence of one or more attributes assigned to a relationship may indicate that the relationship conceals an unidentified entity. For example, the presence, of the Date_View and Comments attributes on the Views relationship may indicate the presence of an entity called Viewing.

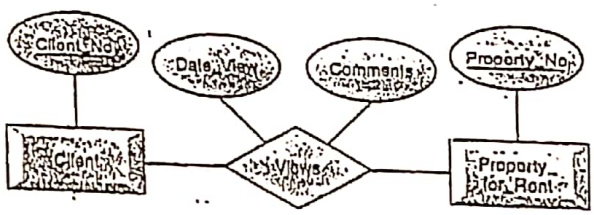


Figure Lecture5_5.10

Structural Constraints

We now examine the constraints that may be placed on participating entities in relationship. The constraints should reflect the restrictions on the relationships a perceived in the 'real world'. Examples of such constraints include the requirements that a property for rent must have an owner and each branch office must be allocated staff. There are two main types of restriction on relationships called cardinality and participation constraints.

Cardinality Constraints

Cardinality ratio	Describes the number of possible relationships for each participating entity.

The most common degree for relationships is binary and the cardinality ratios of binary relationships are one-to-one (1:1), one-to-many (1:M), and many-to-mall (M:N).
The cardinality ratio between entities is a function of the policies established by an enterprise. The rules defining cardinality are referred to as business rules. Ensuring that all appropriate business rules are identified and represented is an important part of modeling an enterprise. Unfortunately, not all business rules can be represented in an ER model. An example of such a business rule is the requirement that a member of staff receives an additional day's holiday for every year of employment with the enterprise.

One-to-one relation ship

Consider the binary relationship Manages, which relates the Staff and Branch entities, Figure Lecture5_5.11(a) represents the Staff Manages Branch relationship using a semantic net model; Note that, to simplify the semantic net models shown in this section, only some of the attributes associated with each entity are shown.

The semantic net model shown in Figure 5.11(a) displays individual occurrences of the Manages relationship between the Staff and Branch entities, for example, Susan Brand (s1) is the Manager of branch office B3 (b1) in Glasgow and John White (s3) is the Manager of branch office 85 (b2) in London.

From Figure 5.11(a), we also note that Ann Beech (s2) is not a Manager, and is therefore not associated with the Manages relationship. However, in determining the cardinality ratio of a relationship, we are interested only in entities that are involved in the relationship. The involvement of each entity in a given relationship is called the 'entity participation'.

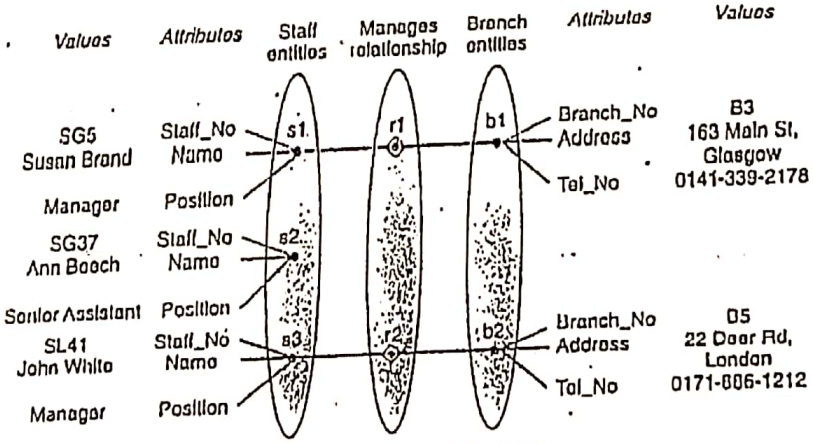


Figure Lecture5_5.11(a)

We note from the semantic net diagram of the Manages relationship that a single Staff entity (Manager) is associated with a single Branch entity (branch office), and therefore the Manages relationship is a one-to-one (I: I) relationship. In other words, the cardinality ratio for the Manages relationship is 1:1. We confirm the cardinality of this relationship using the business rule that it represents.

An ER diagram of the Staff Manages Branch relationship is shown in Figure Lecture5_5.11 (b). In general, the participants in each relationship are connected by lines, which are labeled with 1, M, or N as determined by the cardinality ratio of the relationship.

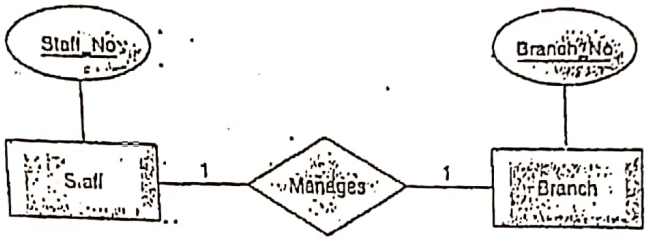


Figure Lecture5_5.11(b)

Consider the binary relationship Oversees, which relates the Staff and Property_For_Rent entities. Figure Lecture5_5.12(a) represents the Staff *Oversees* Property_for_Rent relationship using a semantic net model.

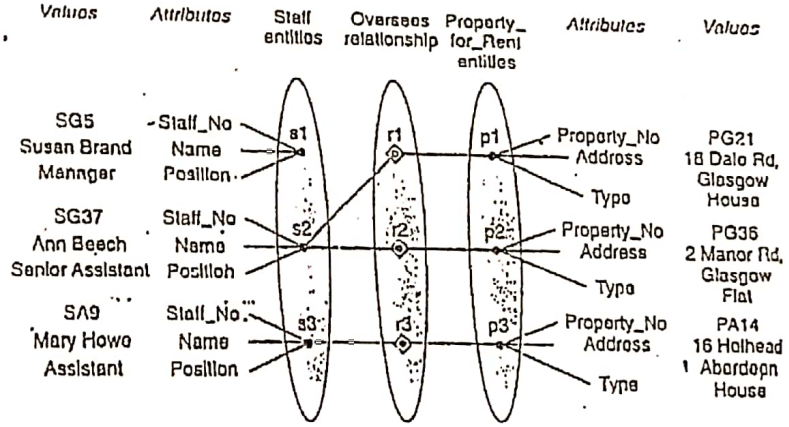


Figure Lecture5_5.12(a)

This diagram displays individual occurrences of the Oversees- relationship between the Starr and Property_for_Rent entities. For example, Ann Beech (s2) manages two

properties in Glasgow PG21 and PG36 (p1 and p2), and Mary Howe (s3) manages a single property PA14 (p3) in Aberdeen. Susan Brand (s1) is not involved in the Oversees relationship. As we stated above, in determining the cardinality ratio of a relationship, we are interested only in entities that are specifically involved in the relationship. We note that a single Staff entity can be associated with one or more Property_for_Rent entities. and therefore the Oversees relationship from the viewpoint of the Staff entity is a one-to-many (1:M) relationship.

If we examine the Oversees relationship from the opposite direction, we note that property numbers PG21 (p1) and PG36 (p2) located in Glasgow are managed by Ann Beech (p2). Property number PA14 (p3) in Aberdeen is managed by Mary Howe (s3). We note that a single Property_for_Rent entity is associated with a single Staff entity, and therefore the Oversees relationship from the view, point of the Property_For_Rent entity is a one-to-one (I: I) relationship.

In summary the Oversees relationship is 1 : M from the viewpoint of the Staff entity and 1:1 from the viewpoint of the Property_For_Rent entity. However we represent the relationship using the higher cardinality that is from the view point of the Staff entity. In other words the cardinality ratio for the Oversees relationships is 1: M. An ER diagram of the Staff Oversees Property_For_Rent relationship is shown in Figure Lecture5_5.12(b). We confirm the cardinality of this relationship using the business rule that it represents.

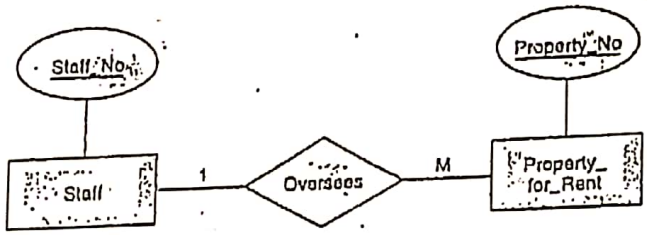


Figure Lecture5_5.12(b)

Consider the binary relationship Advertises, which relates the Newspaper and Property_for_Rent entities. Figure Lecture5_5.13(a) represents the Newspaper Advertises Property _for_Rent relationship using a semantic net model.

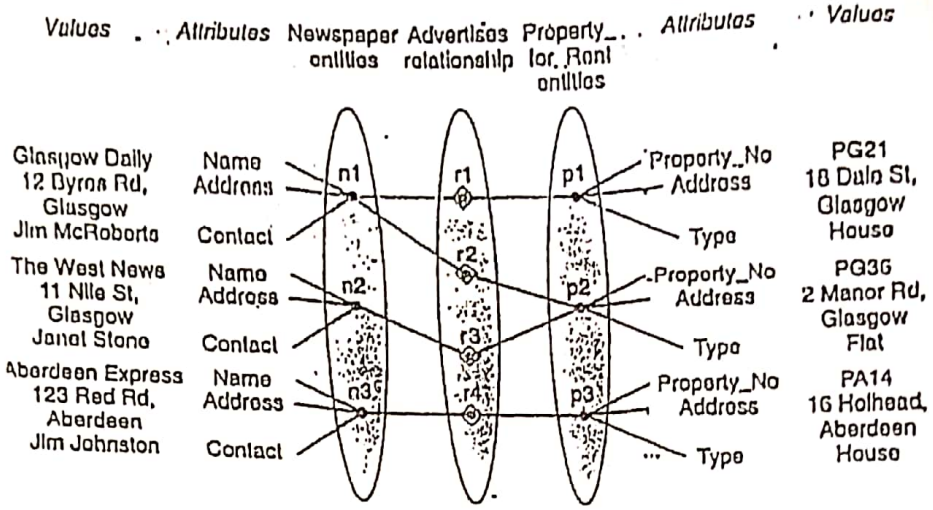


Figure Lecture5_5.13(a)

This diagram displays individual occurrences of the Advertises relationship between the Newspaper and Property_for_Rent entities. For example the Glasgow Daily advertises two properties PG_1 and PG36 (p1 and p2). The West News advertises a single property PG36 (p2), and the Aberdeen Express advertises a single property PA14 (p3). We note that a single Newspaper entity can be associated with one or more Property_for_Rent entities and therefore the Advertises relationship from the viewpoint of the Newspaper entity is a one-to-many (1 : M) relationship.

If we examine the Advertises relationship from the opposite direction, we note that property number PG36 (p2) is advertised in the Glasgow Daily and The West News (n1 and n2). We conclude that a single Property_for_Rent entity can be associated with one or more Newspaper entities, and therefore the Advertises relationship from the viewpoint of the Property_for_Rent entity is a one-to-many (1 :M) relationship.

In summary, the Advertises relationship is 1:M from the viewpoint of both the Newspaper and Property_for_Rent entities. We represent this relationship as two one-to-many relationships in both directions, which are collectively referred to as a many-to-many (M:N) relationship. In other words, the cardinality ratio for the Advertises relationships is M:N. An ER diagram of the Newspaper Advertises Property_for_Rent relationship type is shown in Figure Lecture5_5.13(b). We confirm the cardinality this relationship using the business role that it represents.

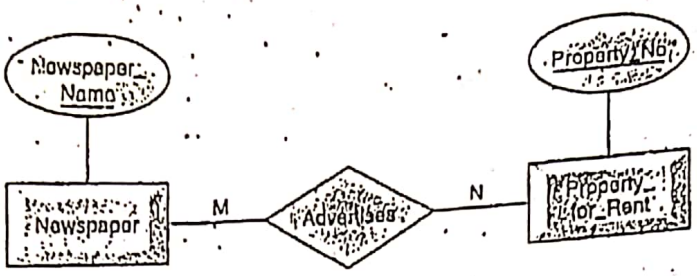


Figure Lecture5_5.13(b)

Participation Constraints

Participation constraints Determines whether the existence of an entity depends upon it being related to another entity through the relationship.

There are two types of participation constraints, total and partial. The participation is total if an entity's existence requires the existence of an associated entity in a particular relationship, otherwise the participation is partial. For example in the, Branch IsAllocated Staff relationship, if every branch office is allocated members of staff, then the participation of the Branch entity in the IsAllocated relationship is total. However, if some members of staff (for example, Sales Personnel) do not work at a particular branch office, then the participation of the Staff entity in the IsAllocated relationship is partial.

The representation of the participation constraints associated with the Branch IsAllocated Staff relationship is shown in Figure Lecture5_5.14. The terms total and partial participation are sometimes referred to as mandatory and optional participation. The participants in each relationship are connected by lines, which are single if the participation partial and double if the participation is total.

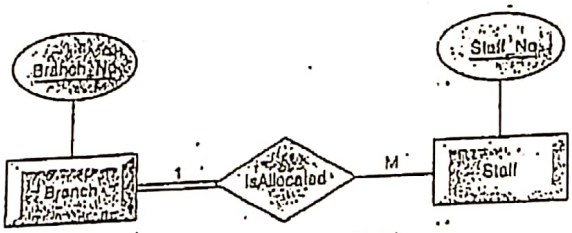


Figure Lecture5_5.14

We may use an alternative notion for displaying the structural constraints of a relationship by displaying the minimum and maximum (Min, Max) values next to the connecting line that represents the participation of the entity in the relationship. For example, we use this notation to represent the structural constraints associated with the Branch IsAllocated Staff relationship in Figure 5.15. The usefulness of this notation is that sometimes more information on the constraints of the relationship is displayed. For

example, in Figure 5.15 the (5,N) notation between the Branch entity and the IsAllocated relationship indicates that there is a minimum of five members of staff (Min = 5) working at each branch office and an unspecified maximum number (Max = N). Similarly, the (0,1) notation between the Staff entity and the IsAllocated relationship means that a member of staff need not work at any particular branch office (Min = 0) or a member of staff may work at a maximum of one branch office (Max = 1). This information is not represented if the simpler values representing cardinality are used, namely 1, M, or N. Another way of indicating the participation constraints is as below

Relationship Participation

A participating entity in a relationship is either optional or mandatory; The participation is optional if one entity occurrence does not require a corresponding entity occurrence in a particular relationship. For example, suppose that College employs some professors who conduct research without teaching classes. If we examine the relationship PROFESSOR teaches CLASS, it is quite possible for a PROFESSOR not to teach a CLASS. Therefore, CLASS is optional to PROFESSOR. On the other hand, a CLASS must be taught by a PROFESSOR. Therefore, PROFESSOR is mandatory to CLASS.

An optional entity is shown by drawing a small circle (o) on the side of the optional entity as shown in figure Lecture5_4.15

Figure Lecture5_4.15

Note also the cardinality rules displayed in Figure Lecture5_4.15, A PROFESSOR may teach no classes at all; or may teach as many as three classes. On the other hand, a class is taught by one (and only one) professor:

Failure to understand the distinction between mandatory and optional participation in relationships may yield designs in which awkward (and unnecessary) temporary entities must be created just to accommodate the creation of required entities. Therefore, it is important to understand the mandatory and optional concepts clearly.

It is also important to understand that the semantics of a problem may determine the type of participation in a relationship. For example, suppose that College offers several courses; each course has several classes. Note again the distinction between class and course in this discussion: A CLASS constitutes a specific offering (or section) of a COURSE. Analyzing the CLASS entity's contribution to the "COURSE generates

CLASS" relationship, it is easy to see that CLASS is existence-dependent on COURSE because a course's section (CLASS) cannot exist without that course. Therefore, we conclude that the COURSE entity is mandatory in the relationship. But we may write two scenarios for the COURSE entity shown in Figures Lecture5_4.16 and Lecture5_4.17.The different scenarios are a function of the semantics of the problem; that is, they depend on the business rules that govern the relationship.

1). CLASS is optional. It is possible for the department to create the entity COURSE first and then create the CLASS entity after making the teaching assignments. In the real world such a scenario is likely; there may be courses for which sections (classes) have not yet been defined. In fact, some courses are taught only once a year and do not generate classes each semester:

2). CLASS is mandatory. This condition is created by the constraint that is imposed by the semantics of the statement, "Each COURSE has one or more Classes. In E-R terms, each COURSE in the "generates" relationship must have at least one CLASS. Therefore, a CLASS must be created as the COURSE is created, in order to comply with the semantics of the problem.

Figure Lecture5_4.16

Figure Lecture5_4.17

Keep in mind the practical aspects of the scenario presented in Figure Lecture5_4.17. The system will not accept a course that is not associated with at least one section. Is such a rigid environment desirable from an operational point of view? For example, when we create a new COURSE, the database first updates the COURSE table, thereby inserting a COURSE entity that does not yet have a CLASS associated with it. Naturally, the problem seems to be solved when CLASS entities are inserted into the corresponding CLASS table. However; given the semantics of the mandatory relationship, the system will be in temporary violation of the business rule constraint. A COURSE deletion similarly causes temporary data integrity problem. For practical purposes, it would be desirable to classify the CLASS as optional in order to produce a more flexible design.

Prepared and Compiled by
S. M. KHALID JAMAL

As you examine these scenarios, keep the role of the DBMS in mind. To maintain data integrity, the DBMS must assure that the "many" side (CLASS) is associated with a COURSE E through the foreign key rules. The burden of maintaining data integrity is assigned to the many side rather than to the one side. The DBMS handles the data integrity constraints imposed by the business rules. (A few DBMS implementations, such as Microsoft's SQL Server and ORACLE, allow the definition of business rules at the DBMS level.)

Remember that the terms mandatory and optional refer to the participation of an entity within the context of a relationship with another entity. The term optional refers to a condition in which the other participating entity may or may not be associated with occurrences of the optional entity in the relationship. In contrast, the term mandatory refers to a condition in which one participating entity must be associated with one or more occurrences of the other participating entity in the relationship.

EXISTENCE DEPENDENCY

An existence constraint or existence dependency can occur between two entities. If X and Y are entities and each instance of Y must have a corresponding instance of X, we say that Y is existence dependent on X. This means a Y entity cannot exist without some X entity. A Y cannot enter the database unless its corresponding X is there, and if the X is dropped from the database, the Y must be dropped as well. X is referred to as strong, parent, owner or dominant entity and Y as the weak, child, dependent or subordinate entity. Another way of expressing existence dependency is to say that the relationship is total mapping from set Y to set X. That means if y in set Y exists, it must be related to some x in the set X.

Existence Dependency: If an entity's existence depends on the existence of one or more other entities, it is said to be existence-dependent.

For example, if you examine Database Table Lecture5_4.4's 1:M relationship "COURSE generates CLASS; you see that the CLASS table's foreign key (CRS_CODE) references the COURSE table. Clearly; if the COURSE does not exist, having the CLASS table's foreign key (CRS_CODE) reference a nonexisting table will generate referential integrity violations. Similarly; if a CLASS must be taught by a PROFESSOR, CLASS is existence-dependent on PROFESSOR in the 1: M relationship "PROFESSOR teaches CLASS:" (One would assume that a class would be canceled if there were no professor to teach it.)

As you examine the data in Database Table Lecture5_4.4, keep in mind that existence dependency means that the order in which the tables are created and loaded is very important. Because CLASS is existence-dependent on COURSE, the COURSE table must be created before the CLASS table. After all, it would not be acceptable to have the CLASS table's foreign key reference a COURSE table that does not yet exist. In some

DBMSes, this sequencing problem does not crop up until the data are loaded into the tables.

Table name: COURSE

COURSE_CODE	DEPT_CODE	DESCRIPTION	CRS_CREDIT
ACCT-211	ACCT	Accounting I	3
ACCT-212	ACCT	Accounting II	3
ACCT-311	ACCT	Managerial Accounting	4
ART-210	ART	Intro. to Art	3
ART-340	ART	Jewelry Design	4
BIOL-120	BIOL	Intro. to Biology	4
BIOL-220	BIOL	Biology and the Environment	3
CIS-220	CIS	Intro. to Microcomputing	3
CIS-320	CIS	Spreadsheet Applications	3
CIS-370	CIS	Intro. to Systems Analysis	4
CIS-420	CIS	Database Design and Implementation	3
ECON-240	ECON/FIN	Macroeconomics	3
ECON-250	ECON/FIN	Microeconomics	3
ENG-210	ENG	Writing	3
ENG-220	ENG	Literature	3
FIN-300	ECON/FIN	Money and Banking	3
HIST-210	HIST	U.S. History Through the 1800s	3
HIST-220	HIST	U.S. History Through the 1900s	3
MATH-120	MATH	College Algebra	4
MATH-240	MATH	Intro. to Calculus	4
MATH-242	MATH	Mathematics for Managers	3

Table name: CLASS

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	EMP_NUM
10012	ACCT-211	1	MWF 8:00-8:50 a.m.	KLR 225	105
10013	ACCT-211	2	MWF 9:00-9:50 a.m.	KLR 225	342
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	KLR 225	301
10015	ACCT-212	1	MWF 10:00-10:50 a.m.	KLR 240	301
10016	ACCT-212	2	Th 6:00-8:40 p.m.	KLR 240	114
10017	ACCT-311	1	TTh 3:30-4:45 p.m.	KLR 240	435
12001	ART-210	1	MWF 8:00-8:50 a.m.	BBG 120	435
12002	ART-340	1	MWF 10:00-10:50 a.m.	BBG 143	110
15020	BIOL-120	1	TTh 12:30-1:45 p.m.	AAK 166	110
15021	BIOL-120	2	Tue 6:00-8:40 p.m.	AAK 156	307
15022	BIOL-120	3	MWF 1:00-1:50 p.m.	AAK 155	307
15030	BIOL-220	1	MWF 2:00-2:50 p.m.	AAK 172	228
20017	CIS-220	1	MWF 9:00-9:50 a.m.	KLR 209	162
20018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR 211	228
20019	CIS-220	3	MWF 10:00-10:50 a.m.	KLR 209	228
20025	CIS-320	1	MWF 10:00-10:50 a.m.	KLR 211	209
20030	CIS-370	1	MWF 11:00-11:50 a.m.	KLR 209	209
20031	CIS-370	2	Tue 6:00-8:40 p.m.	KLR 211	162
20040	CIS-420	1	Wed 6:00-8:40 p.m.	KLR 209	299
22010	ECON-240	1	MWF 8:00-8:50 a.m.	KLR 290	425
22011	ECON-240	2	TTh 3:30-4:45 p.m.	KLR 290	

Database Table Lecture5_4.4

There are basically two major types of existence dependency, they are

1) Identifier dependency
2) Referential dependency

Identifier Dependency

A special type of existence dependency, called and identifier dependency, occurs when the weak entity set does not have a candidate key, and its instances are indistinguishable

without a relationship with another entity. An example of this can be given by Employee and ACR (Annual Confidential Reports). Lets' assume that evaluations are being kept on all employees of the organization. An evaluation entity instance has attributes Date, Head_Of_Department, and Evaluation_Marks. There may be several instances with identical values for all three attributes, so such an entity must be associated with the correct employee instance to have meaning. Figure Lecture5_5.10 shows the E-R diagram with the identifier dependency of Evaluation on Employee. Some authors distinguish identifier dependencies from other existence dependencies by placing "ID" in the relationship diamond above the relationship name, replacing the "E" in the corresponding location for an existence dependency.

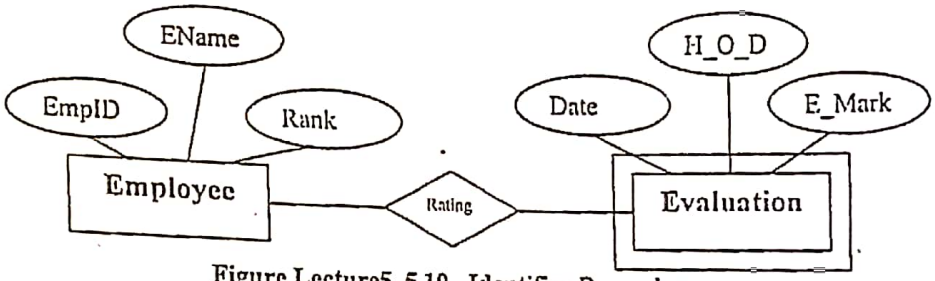


Figure Lecture5_5.10 Identifier Dependency

Referential Dependency

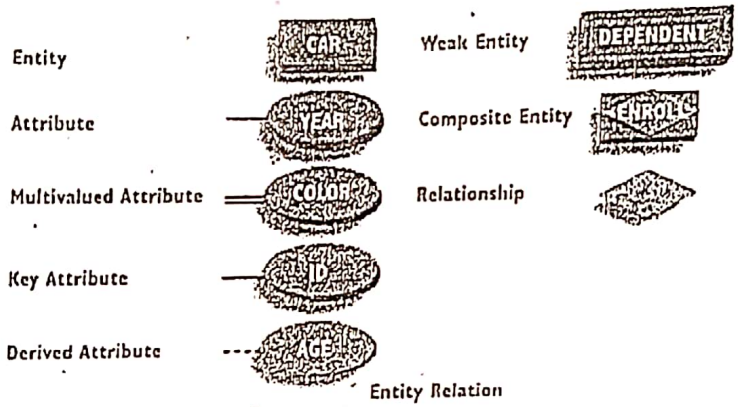
Another special type of existence dependency, called a referential dependency, occurs when a weak entity contains as a foreign key the primary key of the corresponding strong entity. This leads to an important type of constraint called the referential constraint, which states that a non null value of the foreign key attribute in the weak entity instance must always equal the value of the primary key of an associated strong entity instance. For example, if the Dept_Name appears as an attribute of the Employee entity and Employee entity is existence dependent on Department, then Dept_Name is the foreign key for the Employee entity set we are assuming, of course, that Dept_Name is the key of the Department entity. The constraint says that in every Employee entity instance the value of Dept_Name must match the value of Dept_Name in some Department entity instance. Figure Lecture5_5.10 shows that this is a referential dependency by placing an "R" above the relationship name in the relationship dimond.

had been defined as a composite key): composed of the combination CRS_CODE and CLASS_SECTION. we may represent CLASS by:

CLASS, (CRS_CODE, CLASS_SECTION, CLASS_TIME, CLASS_ROOM, PROF_NUM)

In this case, the CLASS primary key is partially derived from COURSE, because the CRS_CODE is the COURSE's primary key. Given this decision, CLASS is a weak entity by definition. In any case, CLASS is always existence-dependent on COURSE, whether or not it is defined to be weak!

Weak entities and existence dependency are related to each other in the sense that the Weak entities are existence dependent on other entities.



Entity Relation

Generalization Hierarchy

Problems with ER Models

In this section, we examine several problems that may arise when designing a conceptual data model. These problems are referred to as connection traps, and normally occur due to a misinterpretation of the meaning of certain relationships. We examine two main types of connection traps, called Fan traps and chasm traps and illustrate how to identify and resolve such problems in ER models, However, it is worth mentioning that although it is important to check a data model for potential connection traps, some of those found may not be significant to the enterprise whilst others are, and require the restructuring of the conceptual model.

In general to identify connection traps we must ensure that the meaning of a relationship is fully understood and clearly defined. If we do not understand the relationships we may create a model that is not a true representation of the 'real world'.

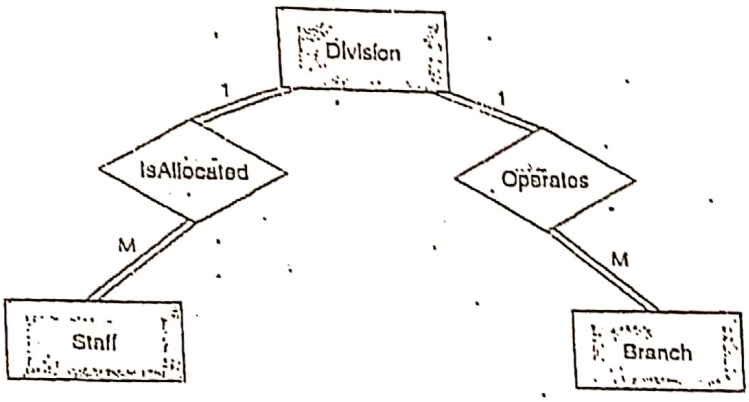


Figure Lecture5_16(a) E.g. of Fan Trap

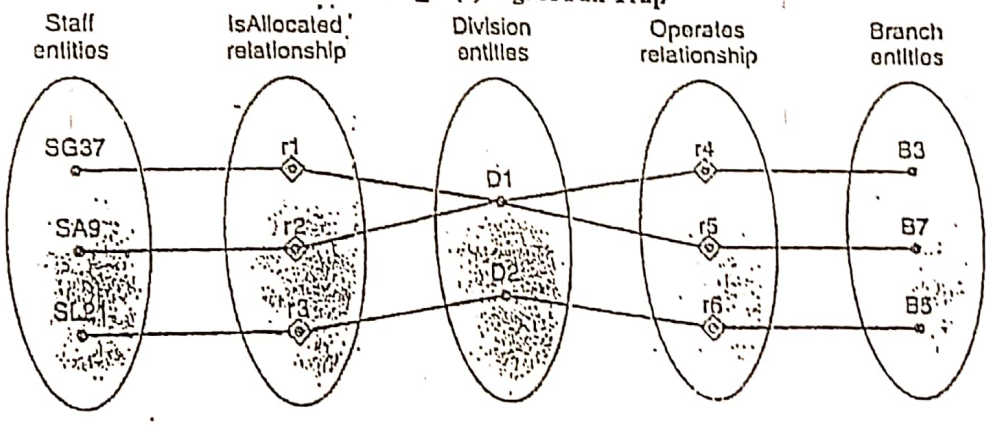


Figure Lecture5_5.16(b) Semantic Net model of ER in fig 5.16(a)

Fan Trap

Fan trap Where a model represents a relationship between entity types, but the pathway between certain entity occurrences is ambiguous.

A fun trap may exist where two or more 1:M relationships fan out from the same entity. A potential fun trap is illustrated in Figure Lecture5_5.16(a), which shows two 1 : M relationships (IsAllocated and Operates) emanating from the same entity called Division.

We conclude from the ER model shown in Figure Lecture5_5.16(a) that a single division operates many branch offices and is allocated many staff. However, a problem arises when we want to know which members of staff work at a particular branch office. To appreciate the problem, we examine the ER model shown in Figure Lecture5_5.16(a) at the level of individual occurrences, using the semantic net model shown in Figure 5.16(b)

Using the semantic net model, we attempt to answer the following question:

- At which branch office does staff number SG37 work?.

Unfortunately, with the current structure it is impossible to give a specific answer. From the semantic net model shown in figure Lecture5_5.16 (b), we can only determine that staff number SG37 works at Branch B3 or B7. The inability to answer this question specially is the result of fan trap associated with the misrepresentation or the current relationships between the Staff, Division and Branch entities. We can resolve this fan trap by restructuring the original ER model to represent the correct association between these entities. As shown in Figure. 5.17(a).

If we now examine this structure at the level of individual occurrence as shown in Figure Lecture5_5.17 (b), we can see that we are now in a position to answer the type of question posed earlier. From this semantic net model, we can determine that staff number SG37 works at branch office number B3, which is part of the division D1.

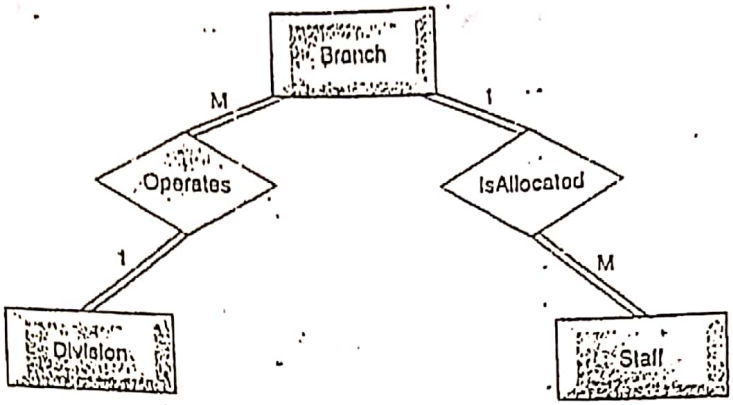


Figure Lecture5_5.17(a) restructured ER model of 5.16(a)

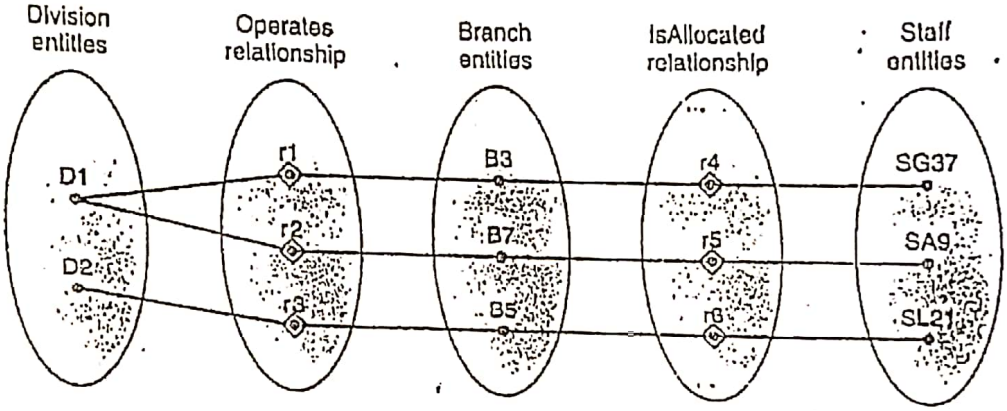


Figure Lecture5_5.17(b) Semantic Net model of the restructured ER model

Chasm Traps

Chasm trap Where a model suggests the existence of a relationship between entity types but the pathway does not exist between certain entity occurrences.

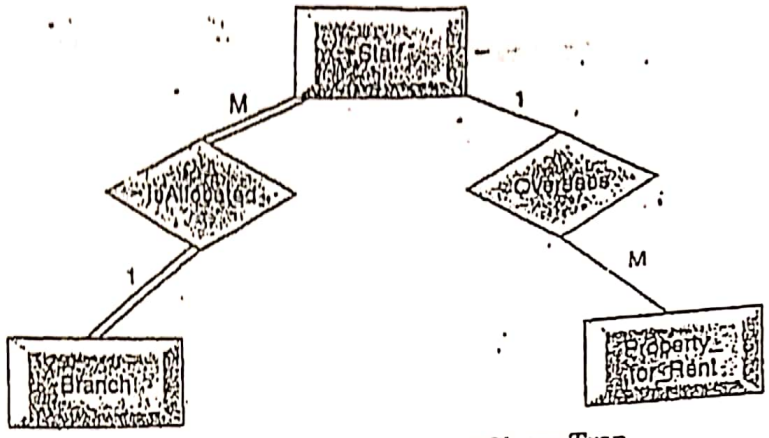


Figure Lecture5_5.18(a) E.g. of Chasm Trap

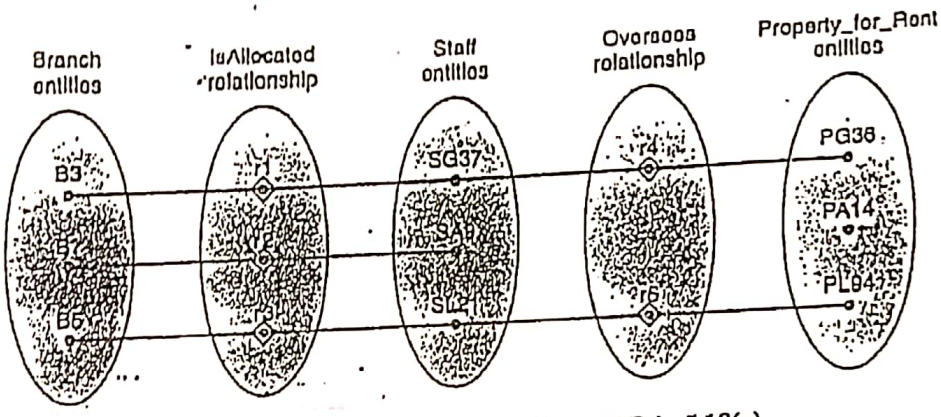


Figure Lecture5_5.18(b) Semantic Net of ER in 5.18(a)

A chasm trap may occur where there is a relationship with partial participation which forms part of the pathway between entities that are related. A potential chasm trap is illustrated in Figure Lecture5_5.18(a), which shows the relationships between the Branch, Staff, and Property_for_Rent entities.

We conclude from this ER model that a single branch is allocated many staff who oversees the management of properties for rent. We also note that not all staff oversees property, and not all properties are managed by a member of staff. A problem arises when we want to know what properties are available at each branch office. To appreciate the problem we examine the ER model shown in Figure Lecture5_5.18(a) at the level of individual occurrences, using the semantic net model shown in Figure Lecture5_5.18(b).

Using this semantic net diagram, we attempt to answer the following question:

- At which branch office is property number P A 14 available?'.

Unfortunately, we are unable to answer this question, as this property is not yet allocated to a member of staff working at a given branch office. The inability to answer this question is considered to be a loss of information (as we know a property must be available at a branch office), and is the result of a chasm trap. The partial participation of Staff and Property_for_Rent in the Oversees relationship means that some properties cannot be associated with a branch office through a member of staff. Therefore to solve this problem, it is necessary to identify the missing relationship, which we call "Has" between the Branch and Property_for_Rent entities. The structure shown in Figure Lecture5_5.19(a) represents the true association between these entities. This structure ensures that, at all times, the properties associated with each branch office are known, including properties that are not yet allocated to a member of staff.

If we now examine this structure at the level of individual occurrences, as shown in Figure Lecture5_5.19(b), we see that we are now in a position to answer the type of question posed earlier. We can now determine that property number PA14 is available at branch number B7.

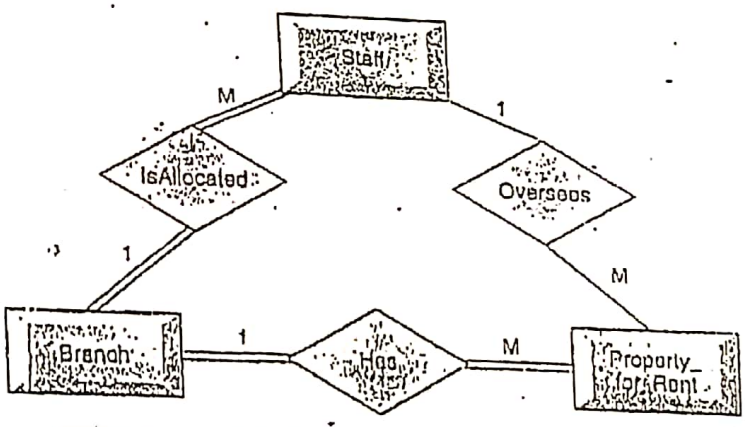


Figure Lecture5_5.19(a) Restructured ER of figure 5.18(a)

There are two important reasons for introducing the concepts or superclasses and subclasses into an ER model. The first reason is that it avoids describing similar concepts more than once, thereby saving time for the designer and making the ER diagram more readable. The second reason is that it adds more semantic information to the design in a form that is familiar to many people. For example, the assertions that 'Manager IS-A member of staff and 'flat IS-A type of property', communicates significant semantic content in a concise form.

Attribute Inheritance

As mentioned above, all entity in a subclass represents the same 'real world' object, as in the superclass, and may possess subclass-specific attributes, as well as those associated with the superclass. For example, the Sales_Personnel subclass has all the attributes of the Staff superclass such as Staff_No, Name, Address, and DOB together with those specifically associated with the Sales_Personnel subclass such as Car Allowance and Sales Area.

A subclass is also an entity, and may therefore also have its own subclasses. An entity and its subclasses and their subclasses, and so on, is called a type hierarchy. Type hierarchies are known by a variety of names including: specialization hierarchy (for example, Manager is a specialization of Staff), generalization hierarchy (for example, Staff is a generalization of Manager), and IS-A hierarchy (for example, Manager IS-A (member of) Staff). We describe the process of specialization and generalization in the following sections.

Specialization

Specialization The process of maximizing the differences between members of an entity by identifying their distinguishing characteristics.

Specialization is a top-down approach to defining a set of superclasses and their, related subclasses. The set of subclasses is defined on the basis of some distinguishing characteristics of the entities in the superclass. When we identify a set of subclasses of an entity type, we then associate attributes specific to each subclass (where necessary), and also identify any relationships between each subclass and other entity types or subclasses (where necessary).

S. M. KHALID JAMAL

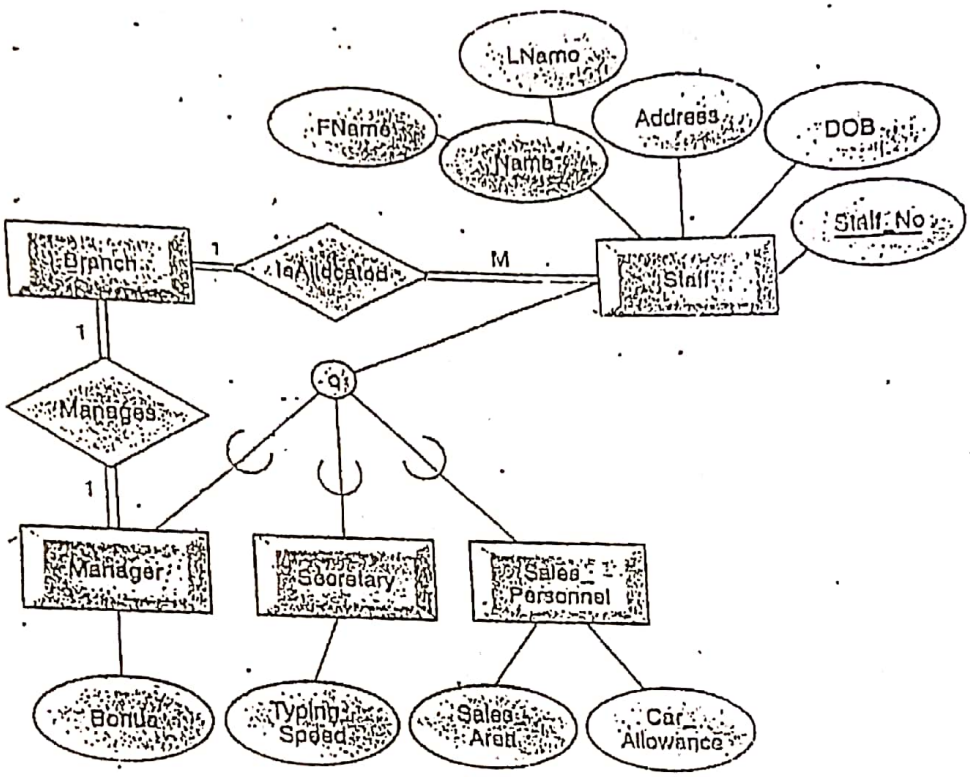


Figure Lecture5_5.20 Specialization of Staff entity into job role subclasses

For example, consider the specialization that identifies the set of subclasses including Manager, Secretary, and Sales_Personnel of the Staff superclass. This specialization can be represented diagrammatically in an EER model, as illustrated in Figure Lecture5_5.20. Note that the Staff superclass and the subclasses, being entity types are represented as rectangles. The subclasses or a specialization are attached by lines to a circle, which is also connected to the superclass. The subset symbol (⊂) on each line that connects a subclass to the circle indicates the direction of the superclass/subclass relationship (for example, Manager (⊂) Staff), The 'o' in the, specialization circle represents a constraint on the superclass / subclass relationship.

Attributes that are only specific to a given subclass are directly attached to the rectangle representing that subclass. For example, the Car_Allowance and Sales Area attributes, shown in Figure Lecture5_5.20, are only associated with the Sales_Personnel subclass, and are not applicable to the manager or the Secretary subclasses. Similarly, we show attributes that are specific to the Manager (Bonus) and Secretary (Typing Speed) subclasses.

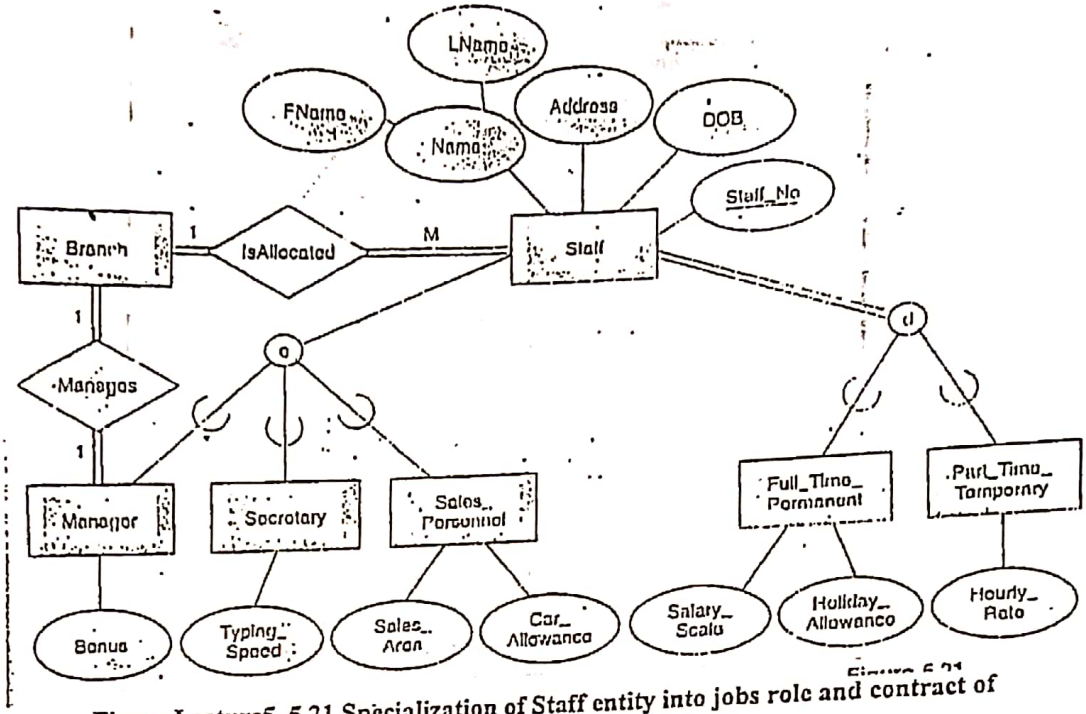


Figure Lecture5_5.21 Specialization of Staff entity into jobs role and contract of employment subclasses

Note that we can also show relationships that are only applicable to specific subclasses. For example, the Manager subclass is related to the Branch entity through the *Manages* relationship, whereas the Staff entity is related to the Branch entity through the *IsAllocated* relationship, as shown in figure Lecture5_5.20.

We may have several specializations of the same entity based on different distinguishing characteristics. For example, another specialization of the Staff entity may produce the subclasses Full_Time_Permanent and Part_Time_Temporary, which distinguishes between the type of employment contract for members of staff. The specialization of the Staff entity type into job role and employment contract subclasses is shown in Figure Lecture5_5.21.

In this figure, we also show attribute that are specific to the Full_Time_Permanent (Salary_Scale and Holiday_Allowance) and Part_Time_Temporary (Hourly_Rate) subclasses. The 'd' in the specialization circle represents a constraint on the superclass/subclass relationship.

A subclass may also have subclasses, which forms a specialization hierarchy. As shown in Figure Lecture5_5.22, Sales_Trainee is a subclass of the Sales_Personnel and Trainee subclasses. A subclass with more than one superclass is called a shared subclass. In other

words, a member of the Sales Trainee shared subclass must be a member of the Sales_Personnel and Trainee subclasses.

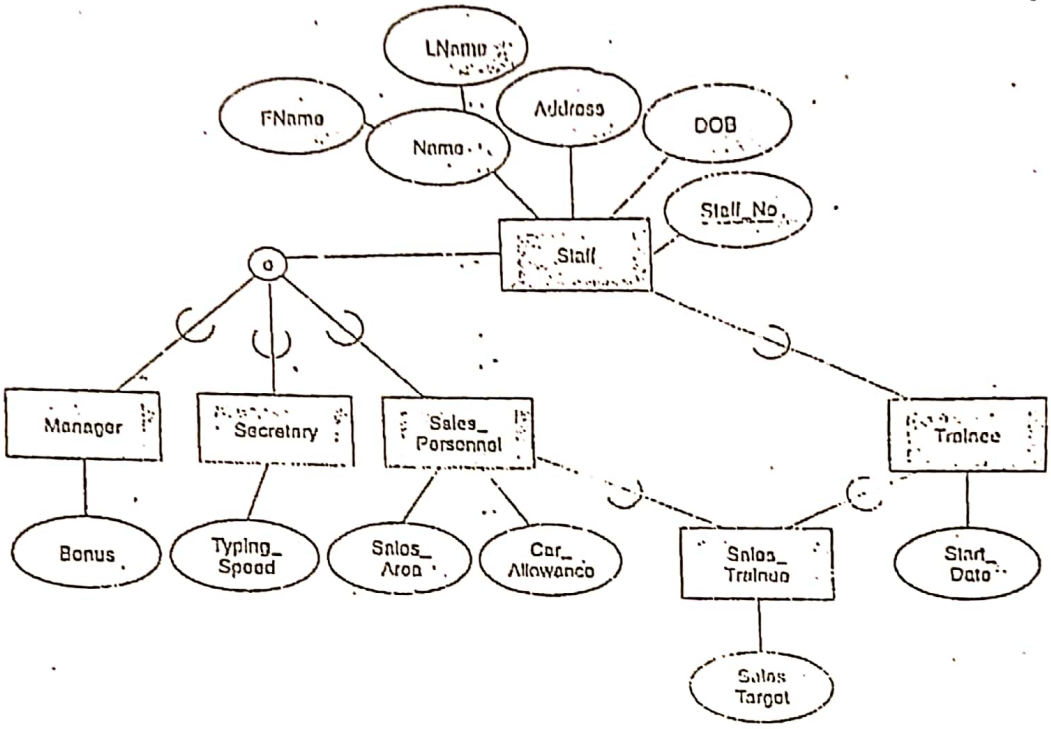


Figure Lecture5_5.22 Sales_Trainee: A shared subclass (multiple inheritence)

As a consequence, the attributes of the Sales_Personnel. (Sales_Area and Car_Allowance) and Trainee (Start_Date) subclasses are inherited by the Sales_Trainee subclass, which also has its own additional attribute called Sales_Target. This process is referred to as multiple inheritance.

Generalization:

Generalization The process of minimizing the differences between entities by identifying their common features.

The process of generalization is a bottom-up approach, which results in the identification of a generalized superclass from the original superclasses. The process of generalization can be viewed as the reverse of the specialization process. For example consider a model where Manager, Secretary, and Sales_Personnel are represented as distinct entities. If we

apply the process of generalization on these entities, We attempt to identify any Similarities between them such as common attributes and relationships. As stated earlier, these entities share attributes, common to all staff, and therefore we would identify Manager, Secretary, and Sales_Personnel as subclasses of a generalized Staff superclass, as previously shown in figure Lecture5_5.20.

Constraints on Specialization and Generalization

We discuss constraints that may apply to a specialization or generalization. Although we only describe these constraints in relation to a specialization, they apply equally to generalization.

The first constraint is called **disjoint** constraint. This constraint specifies that if the subclasses of a specialization are disjoint, then an entity can be a member of only one of the subclasses of the specialization. To represent a disjoint specialization a 'd' for disjoint is placed in the circle that connects the subclasses to the superclass. The subclasses of the contract of employment specialization (Full_Time_Permanent, Part_Time_Temporary) illustrated in Figure Lecture5_5.2(a) are disjoint. This means that a member of staff is either on a full-time permanent contract or a part-time temporary contract.

If subclasses of a 'specialization are not disjoint, then an entity may be a member of more than one subclasses or a specialization. To represent a **non-disjoint** specialization, an 'o' for overlapping is placed in the circle that connects the subclasses to the superclass. The subclasses of the job role specialization (Manager, Secretary, Sales_Personnel) illustrated in Figure Lecture5_5.21 are non-disjoint. In this example, it means that an entity can either be a member or both the Manager and Sales_Personnel subclasses.

The second constraint on a specialization is called the **Participation** constraint, which may be local or partial. A specialization with total participation specifics that every entity in the superclass must be a member of subclass in the specialization. To represent total participation, a double line is drawn between the superclass and the specialization circle, In Figure Lecture5_5.21, the contract of employment specialization has total participation, which means that every member of staff must be either a full-time permanent contract or a part-time temporary contract.

A specialization with partial participation specifies that an entity need not belong to any of the subclasses or a specialization, A partial participation) is represented as a single line between the superclass and the specialization circle in Figure Lecture5_5.21, the job role specialization has partial participation, which means that a member of staff need not have an additional job role such as Manager, Secretary or Sales_Personnel.

The disjoint and participation constraint; of specialization and generalization are distinct. There are four categories as follows; disjoint and total, disjoint and partial, overlapping and total, and overlapping and partial.

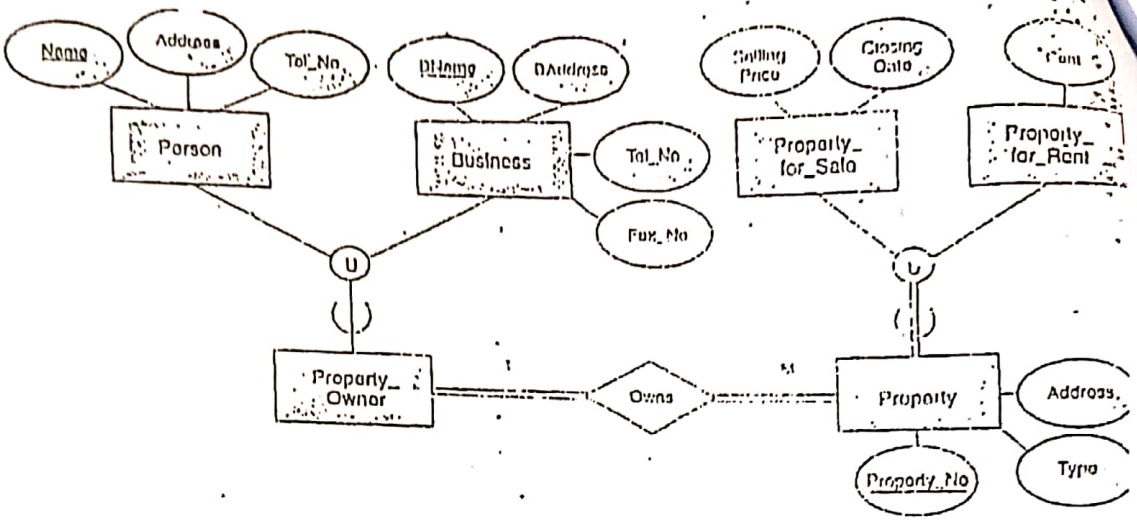


Figure Lecture5_5.23(a) Property_Owner and Property categories

Categorization

Categorization	The modeling of a single subclass with a relationship that involves more than one distinct superclass.

Every superclass/subclass relationship (including those of a shared subclass) in a specialization/generalization hierarchy has a single distinct superclass. For example: the shared subclass called Sales_Trainee in Figure Lecture5_5.22 has two distinct superclass/subclass relationships, where each relationship has a single superclass. However, certain situations require the modeling of a superclass/subclass relationship with more than one distinct superclass. In this case, we call the subclass a category (Elmasri, 1994).

For example two categories called Property_Owner and Property are shown in Figure Lecture5_5.23(a). The Property_Owner category is associated with two distinct superclass entity types, namely Person and Business. The Property category is associated with two superclasses, namely Property_for_Sale and Property_for_Rent. The line connecting the category subclass to the categorization circle has the subset symbol ( ), and the circle itself contains the union symbol (u).

A category subclass has **selective inheritance**, which means for example that each Property_Owner entity inherits only the attributes of the Person superclass (Name, Address, and Tel_No) *or* the attributes of the Business superclass (BName, BAddress, Tel_No, and Fax_No), as shown in Figure Lecture5_5.23(a).

As with specialization and generalization, a category can be further divided based on total or partial participation. For total participation, every occurrence all the superclasses must appear in the category and this is represented by a double line connecting the category subclass to the circle. For partial participation the constraint is removed so that every occurrence of all the superclasses need not appear in the category and this is represented by a single line connecting the category subclass to the circle.

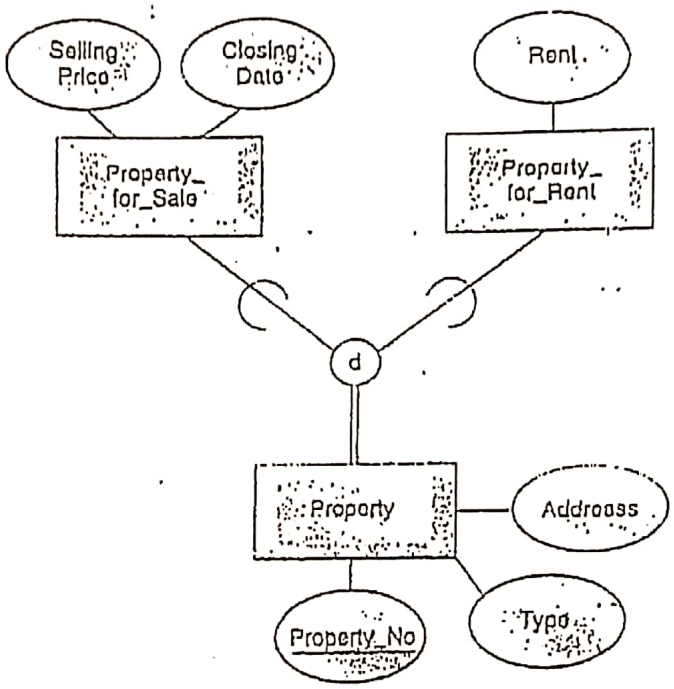


Figure Lecture5_5.23(b) Property specialization/generalization

For example. the Properly _Owner category exhibits partial participation as every occurrence of the Person and the Business superclass need not be represented by the category. On the other hand, the Properly category exhibits, total participation as every member of the Property_for_Sale and the Property_for_Rent superclasses is a member of the category.

When a category has total participation, as is the case for the Property category, there is an option to represent entities as a specialization/generalization. Although this choice is often subjective; it is better to use specialization/generalization when the entities represent the same type of entities, in that they share most attributes, including the primary key. With this in mind, Property _Owner entities are better represented as a category, while the Property entities are best represented as a specialization/generalization, as shown in Figure 5.23(b).

Prepared and Compiled by
S. M. KHALID JAMAL