MODUL - WEEK.12
Enhanced Entity Relationship (ER) Modelling

I. DESKRIPSI TEMA

Design database using structured data model

II. CAPAIAN PEMBELAJARAN MINGGUAN (SUB-CAPAIAN PEMBELAJARAN)

CLO3-SUB-CLO10: Students are able to analyze database (C4)

III. PENUNJANG PRAKTIKUM

1. Tools: Draw.io , yED Graph Editor , Lucidchart
2. Module Practicum.
3. These Module have been adapted from Connolly, T., & Begg, C. (2015). Database Systems: A Practical Approach to Design, Implementation, and Management. 6th edition. Pearson Education. USA. ISBN: 978-1-292-06118-4, Chapter 13.

IV. REVIEW MATERI TEORI

1. Specialization/Generalization

We have discussed different types of relationships that can occur between entities. Some entities have relationships that form a hierarchy. For example, a shipping company can have different types of ships for its business. The relationship that exists between the concept of the ship and the specific types of ships forms a hierarchy. The ship is called a superclass. The specific types of ships are called subclasses.

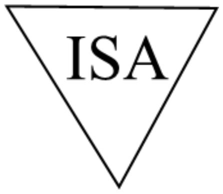
a) **Superclass** : An entity type that represents a general concept at a high level.
b) **Subclass** : An entity type that represents a specific concept at lower levels.

A subclass is said to inherit from a superclass. A subclass can inherit from many superclasses in the hierarchy. When a subclass inherits from one or more super- classes, it inherits all their attributes. In addition to the inherited attributes, a subclass can also define its own specific attributes. A subclass also inherits participation in the relationship sets in which its superclass (higher-level entity) participates.

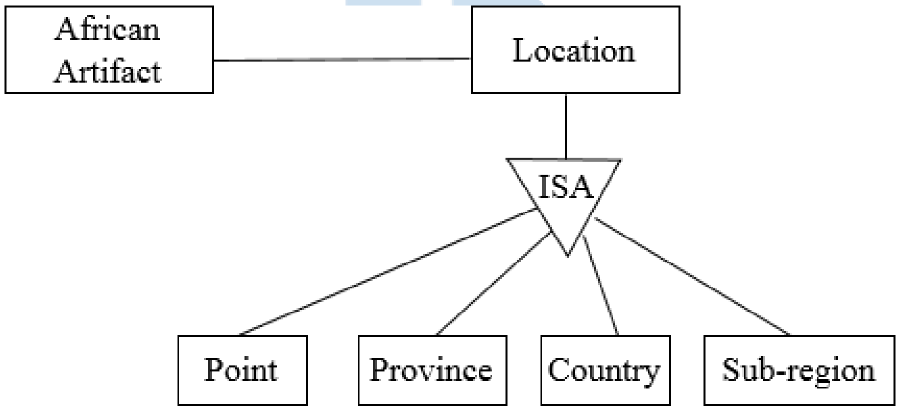
The process of making a superclass from a group of subclasses is called gener- alization. The process of making subclasses from a general concept is called specialization.

a) **Specialization** : A means of identifying sub-groups within an entity set which have attributes that are not shared by all the entities (top-down).
b) **Generalization** : Multiple entity sets are synthesized into a higher-level entity set, based on common features (bottom-up).

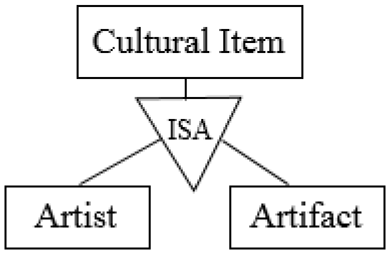
- **Representation of Specialization/Generalization in ER Diagramsz**
  A diamond notation is a common representation of specialization/generalization relationships in ER diagrams.



As an example, let's consider the following scenario:
Africa holds many historical artefacts in different locations. Each artefact is kept in a specific location. A location can be a point, province, country or sub-region of Africa.
The scenario has a specialization relationship between the location and different specific types of locations (i.e. point, province, country and sub-region). This specialization relationship is represented in the ER diagram below.

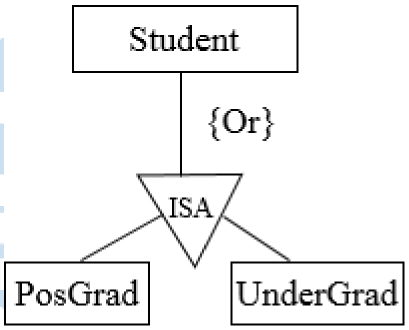


To demonstrate generalization, let's imagine that an Artefact is one of the examples of the African cultural items. Another type of a cultural item is an Artist. It is clear to see that a cultural item is a superclass of an artefact and artist. This generalization relationship can be represented in the ER diagram as show below.

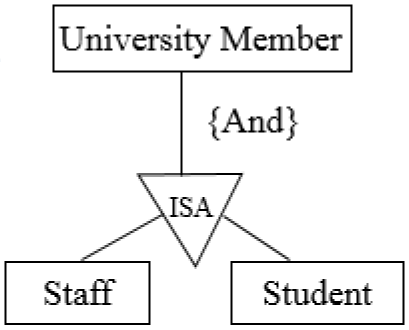


- **Constraints on Specialization/Generalization**
  There are three constraints that may apply to a specialization/generalization: membership constraints, disjoint constraints and completeness constraints.

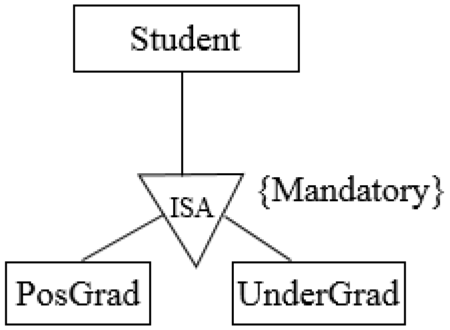
- **Membership constraints**
  Condition defined: Membership of a specialization/generalization relationship can be defined as a condition in the requirements e.g. tanker is a ship where cargo = “oil”
  User defined: Sometimes the designer can define the superclass-subclass relationship. This can be done to simplify the design model or represent a complex relationship that exists between entities.
- **Disjoint constraints**
  Disjoint: The disjoint constraint only applies when a superclass has more than one subclass. If the subclasses are disjoint, then an entity occurrence can be a member of only one of the subclasses, e.g. postgrads or undergrads – you cannot be both. To represent a disjoint superclass/subclass relationship, ‘Or’ is used.



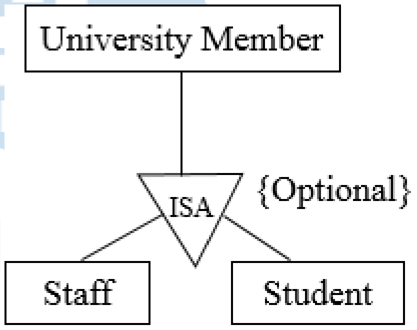
- **Overlapping:** This applies when an entity occurrence may be a member of more than one subclass, e.g. student and staff – some people are both. ‘And’ is used to represent the overlapping specialization/generalization relationship in the ER diagram.



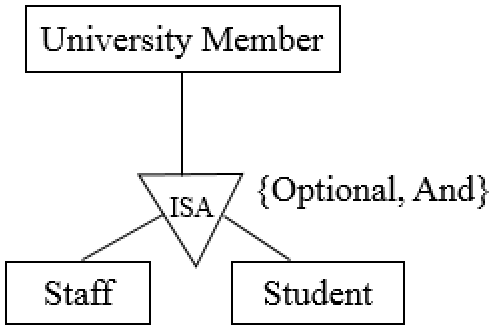
- **Completeness Constraints**
  Total: Each superclass (higher-level entity) must belong to subclasses (lower-level entity sets), e.g. a student must be postgrad or undergrad. To represent completeness in the specialization/generalization relationship, the keyword 'Mandatory' is used.



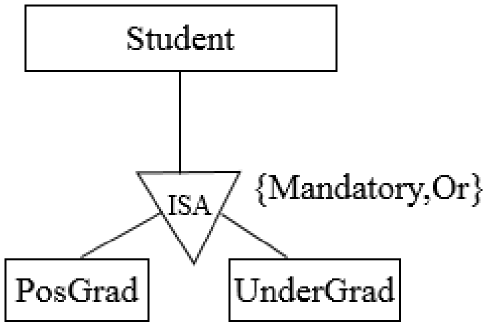
Partial: Some superclasses may not belong to subclasses (lower-level entity sets), e.g. some people at UCT are neither student nor staff. The keyword 'Optional' is used to represent a partial specialization/generalization relationship.



We can show both disjoint and completeness constraints in the ER diagram. Following our examples, we can combine disjoint and completeness constraints.

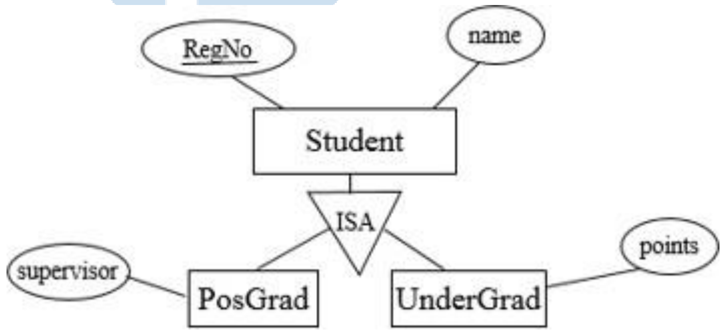


Some members of a university are both students and staff. Not all members of the university are staff and students.



A student in the university must be either an undergraduate or postgraduate, but not both.

- **Mapping Specialization/Generalization to relational tables**
  Specialization/generalization relationship can be mapped to relational tables in three methods. To demonstrate the methods, we will take the student, post-graduate and undergraduate relationship. A student in the university has a registration number and a name. Only postgraduate students have supervisors. Undergraduates accumulates points through their coursework.



Method 1
All the entities in the relationship are mapped to individual tables.
Student (*Regno*, name)
PosGrad (*Regno*, supervisor)
UnderGrad (*Regno*, points)

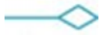
Method 2
Only subclasses are mapped to tables. The attributes in the superclass are duplicated in all subclasses.
PosGrad (*Regno*, name, supervisor)
UnderGrad (*Regno*, name, points)
This method is most preferred when inheritance is disjoint and complete, e.g. every student is either PosGrad or UnderGrad and nobody is both.

Method 3
Only the superclass is mapped to a table. The attributes in the subclasses are taken to the superclass.
Student (*Regno*, name, supervisor, points)
This method will introduce null values. When we insert an undergraduate record in the table, the supervisor column value will be null. In the same way, when we insert a postgraduate record in the table, the points value will be null.

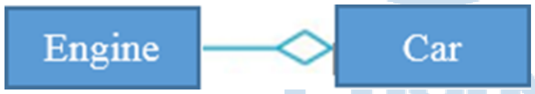
2. Aggregation

Aggregation represents a 'has-a' relationship between entity types, where one represents the 'whole' and the other the 'part'.
An example of aggregation is the Car and Engine entities. A car is made up of an engine. The car is the whole and the engine is the part. Aggregation does not represent strong ownership. This means, a part can exist on its own without the whole. There is no stronger ownership between a car and the engine. An engine of a car can be moved to another car.

- Representation of Aggregation in ER Diagrams
  A line with a diamond at the end is used to represent aggregation.



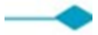
The 'whole' part must be put at the end of the diamond. For example, the Car-Engine relationship would be represented as shown below:



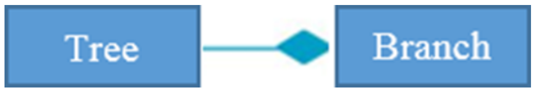
3. Composition

Composition is a form of aggregation that represents an association between entities, where there is a strong ownership between the 'whole' and the 'part'. For example, a tree and a branch have a composition relationship. A branch is 'part' of a 'whole' tree - we cannot cut the branch and add it to another tree.

- Representation of Composition in ER Diagrams
  A line with a **filled** diamond at the end is used to represent composition.



The example of the Tree-Branch relationship can be represented as shown below:



4. XML (eXtensible Markup Language)

In previous chapters, we introduced database technology and how it is used by businesses to store data in a structured format. XML (eXtensible Markup Language) has become a standard for structured data interchange among busi- nesses. It was formally ratified by the World Wide Web Consortium (W3C) in 1998. XML uses markup for formatting plain text. Markup refers to auxiliary information (tags) in the text that give structure and meaning.

We have demonstrated how to use relational tables to represent entities and their attributes. XML also supports the representation of entities and attributes.

In this section, we will introduce XML. Students are encouraged to study de- tailed books for further information. One useful website for learning XML is http://www.w3schools.com/xml/default.asp.

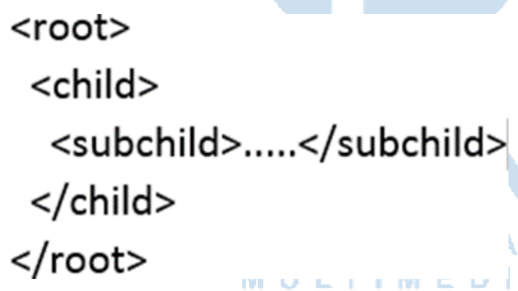
- **Element**

  An element is a building block of an XML document.
  - All elements are delimited by < and >.
  - Element names are case-sensitive and cannot contain spaces.

    The representation of an element is shown below:

    <Element> …. </Element>

    An XML document can contain many elements, but one must be the root element. A root element is a parent element of all other elements.



```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

- **Attribute**

  Elements can have attributes. Attributes are specified by name=value pairs inside the starting tag of an element:

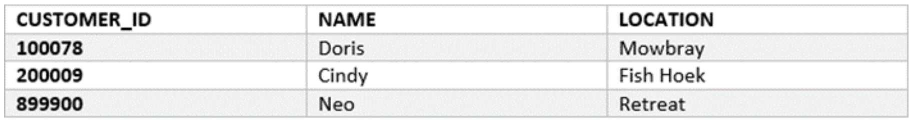
  <Element attribute = “value” >.. </Element >

  All values of the attributes are enclosed in double quotes.

  An element can have several attributes, but each attribute name can only occur once.

  <Element attribute1 = “value1” attribute2=“value2”>

- Example representing relational table records in XML
  To demonstrate XML, let’s imagine we have a customer table that holds information of customers.



CUSTOMER_ID	NAME	LOCATION
100078	Doris	Mowbray
200009	Cindy	Fish Hoek
899900	Neo	Retreat

We can represent the information in XML as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<Customers>

        <Customer customerID=”100078”>

                <Name> Doris <Name>

                <Location> Mowbray </Location>

        </Customer>

        <Customer customerID=”200009”>

                <Name> Cindy <Name>

                <Location> Fish Hoek </Location>

        </Customer>

        <Customer customerID=”899900”>

                <Name> Noe <Name>

                <Location> Retreat </Location>

        </Customer>

</Customers>
```

- **Explanation**
  <?xml version=“1.0” encoding=“UTF-8”?>: is the XML prolog. The prolog is used to specify the version of XML and the encoding used. It is optional, but if it appears in the document, it must be the first line in the document.
  Customers element: Customers is the root element.

Customer element: A Customer element represents a tuple in the Customers table. The table has three attributes, CUSTOMER_ID, NAME and LOCATION. In our XML, CUSTOMER_ID is represented as an attribute of the Customer element. NAME and LOCATION are repre- sented as child elements of the Customer element. Notice that we have repeated the Customer element three times to capture the three records in the Customer table.

- **Document type definition**

The XML technology specifies the syntax for writing well-formed documents but does not impose the structure of the document. XML document writers are free to structure an XML document in any way they want. This can be problematic when verifying a document. How many elements can a document have? What elements should a document have? These questions are difficult to answer unless we also specify the structure of the document. **Document type definition (DTD)** is used to define the structure of an XML document.

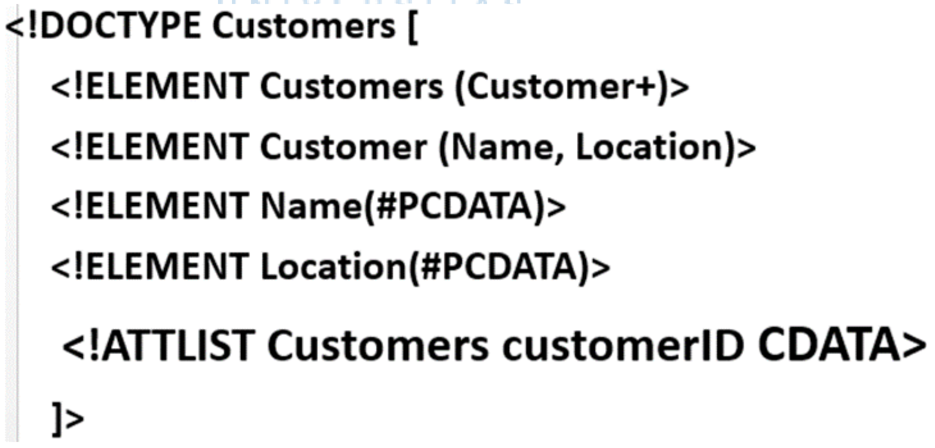
DTD specifies the following:

- What elements can occur.
- What attributes an element can/must have.
- What sub-elements can/must occur inside each element, and how many times.

DTD element syntax:
<!ELEMENT element (subelements-specification) >

DTD attribute syntax:
<!ATTLIST element (attributes) >

The DTD for the XML we defined above can be defined as shown below:



```
<!DOCTYPE Customers [
   <!ELEMENT Customers (Customer+)>
   <!ELEMENT Customer (Name, Location)>
   <!ELEMENT Name(#PCDATA)>
   <!ELEMENT Location(#PCDATA)>
    <!ATTLIST Customers customerID CDATA>
   ]>
```

- **Explanation**
  !DOCTYPE: Defines that the Customers element is the root element of the document.
  <IELEMENT>: Defines an XML element. The first element to be defined is the Customers element. A Customers element has one child element, Customer, indicated in brackets. The + symbol means that the Customer element can appear one or more times under the Customers element. The Customer has two sub-elements, Name and Location. The Name and Location elements have character data as a child element.
  <!ATTLIST>: Defines the attribute. The Customers element has one attribute, customerID, of type character data.

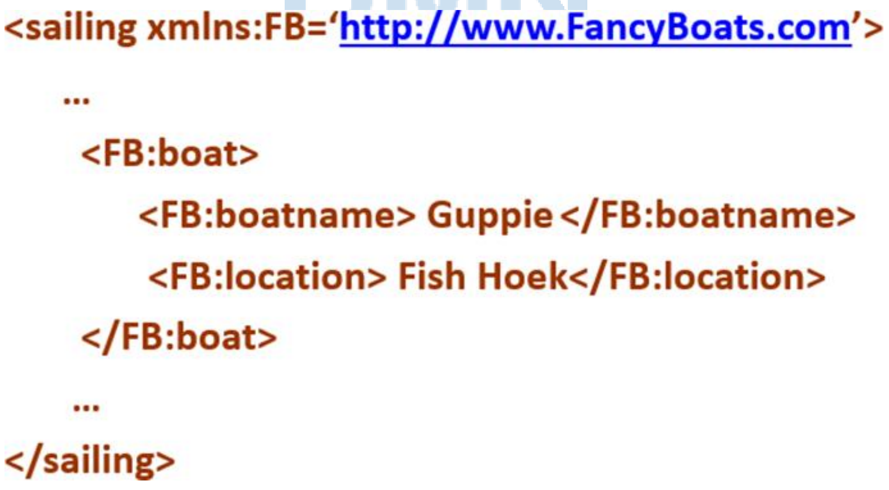
▪ Namespaces

XML data has to be exchanged between organisations. The same element name may have different meaning in different organisations, causing confusion on exchanged documents.

Specifying a unique string as an element name avoids confusion. A better solution is to use a unique name followed by an element name.

unique-name:element-name

Adding a unique name to all element names can be cumbersome for long documents. To avoid using long unique names all over a document, we can use XML namespaces.



```
<sailing xmlns:FB='http://www.FancyBoats.com'>
   ...
    <FB:boat>
        <FB:boatname> Guppie </FB:boatname>
         <FB:location> Fish Hoek</FB:location>
    </FB:boat>
    ...
</sailing>
```

The namespace FB has been declared and initialised to 'http://www.FancyBoats.com'. Namespaces are URIs. URIs are generic identifiers like URLs.

- **xQuery**

  XQuery is a language for finding and extracting elements and attributes from XML documents. The way SQL is to relational databases, XQuery is the query language for XML documents. For example, to display all the names of the customers in the XML above, our XQuery will look as follows:

  for $x in /Customers/Customer
  return $x/Name

V. LATIHAN, STUDI KASUS: Life Insurance Corporation

The Newark divisional office of the Life Insurance Corporation of America keeps all the necessary information about the policy holders in a database. A policy holder pays a premium until the maturity of the policy or his death, at which time the sum assured and bonus is paid to the nominee. The premium to be paid is worked out based on the age of the person proposed and the term of the policy.

Newark division keeps the following information about each policy holder: social security number, name, address, date of birth, description of the terms of the policy and the annual premium.

The corporation has divided its Newark division into 15 zones for its convenience. Each zone has its manager. Every zone has a number of agents allotted, typically ranging from 10 to 20. Every agent must procure a minimum of 10 customers.

Draw an E-R diagram for the Corporation database (state any assumptions you believe you need to make in order to develop a complete diagram). Identify the key attribute (attributes) for each entity and the cardinality of each entity relationship.