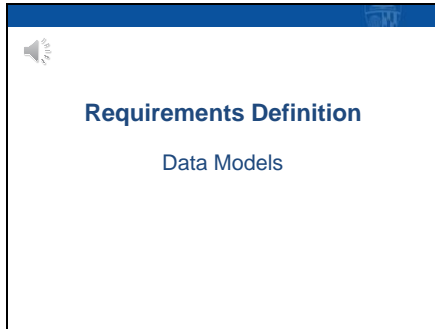
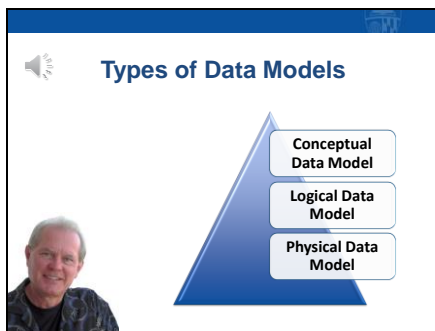


1



In this lecture we'll discuss how to use data models to establish data requirements.

2



Data modeling involves building various kinds of models that describe the data that needs to be stored for a particular software product.

There are three basic kinds of data models that are developed during the course of a software engineering project: a conceptual data model, a logical data model, and a physical data model.

A conceptual data model describes things, called entities, that will be stored, and the relationships between those entities. For example, in a point of sale application for a retail store, information about sale transactions and customers would need to be stored...they would be entities. A relationship that exists between these two entities is that a customer can be associated with many sale transactions, but each sale transaction would be associated with a single customer.

A logical data model elaborates on a conceptual data model by including attributes associated with the entities. In the point-of-sale example, attributes for a sale entity would be things like a sale transaction number, time and date of sale, total sale amount, and so forth. In practice, conceptual data modeling and logical data modeling are often done together.

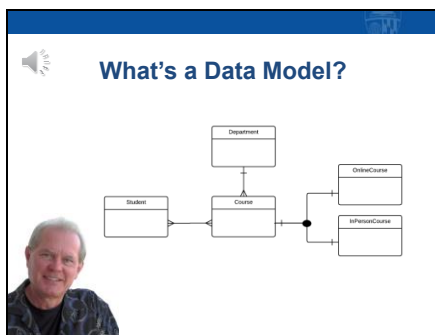
A physical data model describes how the product's database will actually be built. It would describe all the

data tables, including column names, data types, and so forth. This model would typically be developed during the design phase of a project.

Conceptual and logical data models are independent of the particular database software product that will be used. The physical data model will usually be specific to a given database software product, since different database products may use different data types.

In this lecture, I'm going to focus on conceptual and logical data models.

3



So...what's a data model? A data model is an organized representation of business data that will need to be stored and managed. There are a number of different kinds of data models used on software projects...as I mentioned earlier...but we're going to stick to the conceptual and logical data models that are often constructed as part of the requirements phase in a project. I'm going to use the term logical data model in this discussion...since building a conceptual model is a step in the logical data modeling process.

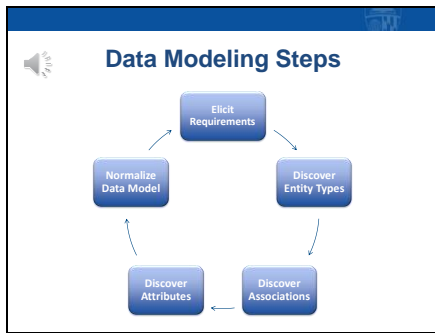
You see a sample data model here. It's not important to understand how to interpret it yet...we'll get there soon. This data model illustrates some of the data types that will be stored and managed as part of a university course management system. Data about departments, courses, and students are illustrated here...as well as the associations or relationships between them. For brevity, I haven't shown the specific data associated with departments, students, and so forth. Oh...by the way...this type of model is also called an entity relationship diagram.

Now, you may be wondering, what the purpose of a data model is. Its purpose is to discover and document data needs and data-related business rules more thoroughly...in other words, it's used to help specify the data requirements for a software product. The data model will ultimately be used to design and implement

efficient data stores for the application under development.

The benefit of using a data model is that it provides more precise, detailed, and unambiguous information about the data that will need to be stored and managed.

4



Data modeling is often an iterative process that starts during the requirements elicitation activity. As needs are elicited, we discover the types of data that will need to be managed and stored.

We call this data entity types. An entity type is a named class of entities which have the same set of descriptive attribute types. As an example, an entity type might be Employee. It will be used to model information that must be stored and managed about individual employees. All employees would have the same set of descriptive attributes, like name address, employee number, hire date...and so on. Each individual employee is called an employee instance. Ultimately, the data for entity types will likely be stored in relational database tables.

We then study the associations or relationships that some of these entity types will have with respect to each other. For example, students will enroll in one or more courses. A student may be associated with a single student account. A university department will offer courses in multiple subjects, and so forth. Technically, a model that shows the entity types and their relationships is called a conceptual data model, as I mentioned earlier in the lecture.

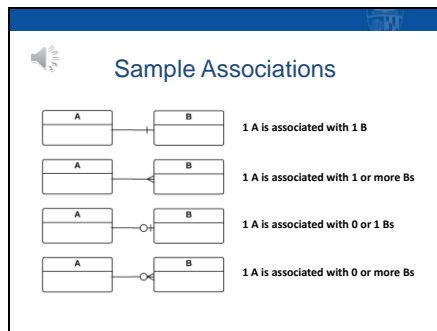
Entity types must have attributes. Attributes correspond to information that will be stored and managed. For example a person entity type will have a

name, a social security number, and lots of other descriptive information that will need to be managed. Attributes typically become fields in a relational database table. When attributes are added to the conceptual model it's technically called a logical data model.

Another step in data modeling, that is usually performed in the later stages, is to “normalize” the logical data model. Normalization involves organizing the data in relational database tables so that the data is not redundant and can be efficiently managed.

As the diagram illustrates, data modeling steps can repeat as more requirements information is elicited and refined.

5



Associations represent relationships that entity types have with one another. I'm showing four basic types of associations: 1-to-1, 1-to-many, 1-to-1 conditional, and 1-to-many conditional. The diagramming notation for each of these associations is illustrated here, using what's called the crow's-foot notation. There are other notations as well that are used in practice.

In a 1-to-1 association, one instance of entity type A is associated with one instance of entity type B. For example, a person is associated with a single driver's license.

In a 1-to-many association, one instance of entity type A is associated with one or more instances of entity type B. For example, a customer could be associated with several sale transactions

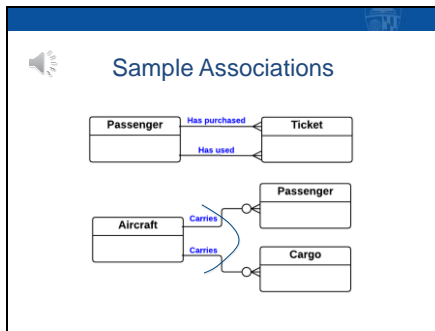
In a 1-to-1 conditional association, one instance of entity type A is associated with zero or one instances of entity type B. For example, a person may have zero or one spouses.

And, in a 1-to-many conditional association, one instance of entity type A is associated with zero or

more instances of entity type B. For example, a policy holder may have filed zero or more claims on an insurance policy.

There are more types of associations than I'm illustrating here, but most of them are formed from these basic four.

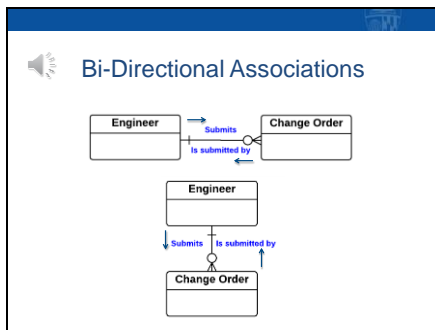
6



We might also have more than one type of association between entity types, as illustrated here between a passenger and a ticket. By the way, in practice we almost always include an association name like in this example, and the association name is a verb.

Some associations are mutually exclusive. For example, due to regulations, an aircraft might carry only cargo or passengers. I've used a little arc in this diagram to indicate the mutual exclusive nature of those associations.

7



Some entity types have bi-directional associations as illustrated here. When the entity relationship diagram is annotated to show this, there are some conventions that are useful to follow.

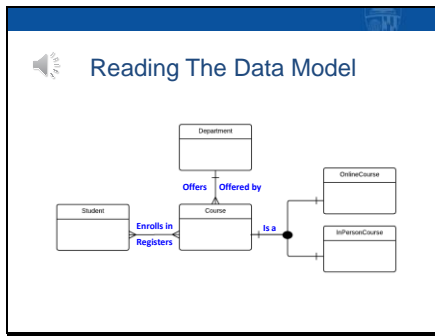
If the entities are laid out left to right, we read that portion of the diagram from left to right, and the association name corresponding to the left-side entity is placed above the connecting link, and the name corresponding to the right-side entity is placed below the link.

If the entities are laid out vertically, we read that portion of the diagram top to bottom, with the association name for the top entity placed to the left of the link and the one for the bottom entity placed to the

right of the link.

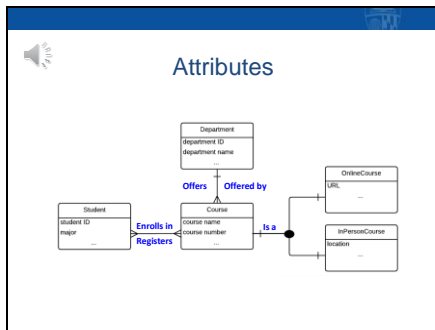
Sometimes, data modelers will only include the left or top association name if the other name is self evident.

8



Let's take a crack at reading this data model. Starting with the department entity...it says that a department offers several courses, but each course is offered by a single department. A student instance enrolls in multiple courses, and a course instance registers multiple students. A course is either an online course or an in-person course. The dot on the diagram is another notation for showing an "or" condition.

9



Attributes are characteristics of entities that provide descriptive detail about them. When they are shown on a data model they are placed in a compartment below the entity name, as illustrated here.

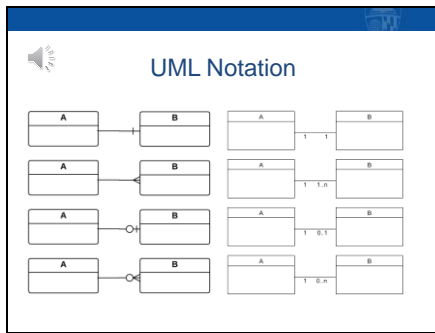
There are two basic kinds of attributes...descriptors and identifiers.

A descriptor attribute specifies a non-unique characteristic of an entity instance. In this diagram, a student's major would be an example of a descriptor attribute, since many students can have the same major.

An identifier attribute...also called a key...is used to uniquely identify an entity instance. Course number

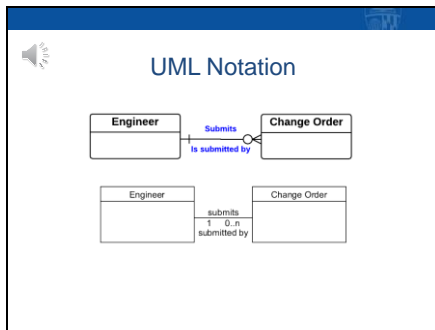
and department number are examples of identifier attributes in this model.

10



So far, I've been using what's called crows foot notation in the diagram examples. The Unified Modeling Language, UML, can also be used to describe data models. I'm showing the crows foot notation for the basic relationships on the left, and the equivalent UML representation on the right.

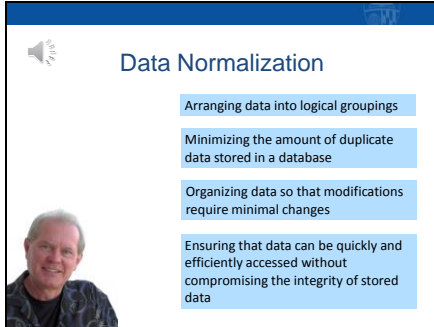
11



Here's a simple relationship using both the crows foot and the UML notation. There are more notations than these two that are used in practice...examples are the Chen notation and Bachman notation...but showing just these two different notations will suffice for our purposes.

In the remainder of this lecture I'll continue to use the crows foot notation.

12



Data Normalization

- Arranging data into logical groupings
- Minimizing the amount of duplicate data stored in a database
- Organizing data so that modifications require minimal changes
- Ensuring that data can be quickly and efficiently accessed without compromising the integrity of stored data

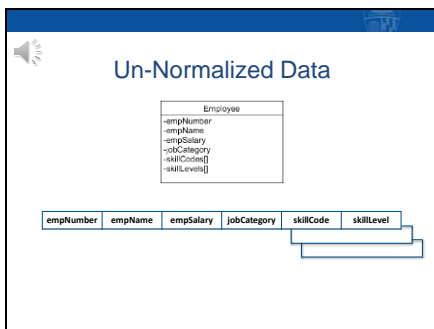
Data normalization is a technique that is used to organize data into tables. Normalization is very important when using relational database systems. It is used as part of logical data modeling for applications that fall into the category of online transactional processing (OLTP)...in which data modifications like inserts, deletes, and updates occur frequently.

When you normalize a database, you have four goals: arranging data into logical groupings such that each group describes a small part of the whole; minimizing the amount of duplicate data stored in a database; organizing the data such that, when you modify it, you make the change in only one place; and building a database in which you can access and manipulate the data quickly and efficiently without compromising the integrity of the data in storage.

Normalization is accomplished by defining a number of relatively simple data tables, and using primary and foreign keys as attributes in the tables to link related information together.

When data modelers discuss normalization, they will often talk about the five normal forms. In this lecture, we'll talk about un-normalized data and the first three normal forms.

13



Un-Normalized Data

Employee

- empNumber
- empName
- empSalary
- jobCategory
- skillCode
- skillLevel

empNumber	empName	empSalary	jobCategory	skillCode	skillLevel

Let's start with data that is un-normalized. Suppose part of the application we are building will require generating reports about different skill sets and levels of experience the employees in an organization have. And, let's suppose the Employee entity illustrated here has been identified during the logical data modeling activity.

For brevity, only a few essential employee attributes are shown here...just enough to focus our discussion on normalization. An Employee entity will have attributes like employee number, employee name, salary, a job category, a list of codes representing various job skills an employee has, and a list of skill levels for each skill,

measured in years.

Think about how the Employee entity data may need to be stored in terms of records of information. Because there can be several skillCode/skillLevel pairs for a single employee, we have what's called repeating groups...as illustrated here.

Data that has repeating groups is not normalized...and can't be handled in relational database systems. Let's see how this data might be normalized.

14

Functional Dependence

B is functionally dependent on A means that if we know the value of A we can find the value of B that is associated with A.

empNumber	empName	empSalary	jobCategory	skillCode	skillLevel
-----------	---------	-----------	-------------	-----------	------------

empName is functionally dependent on empNumber
empSalary is functionally dependent on empNumber
jobCategory is functionally dependent on empNumber
skillLevel is functionally dependent on empNumber + skillCode

Before demonstrating how data can be normalized, it's important to understand the concept of functional dependence. Functional dependence is a very simple concept. If some element B is functionally dependent on an element A, that means that if we know the value of A we can find the value of B that is associated with A.

Let's take the data from the last example. There are a number of functional dependencies in this example.

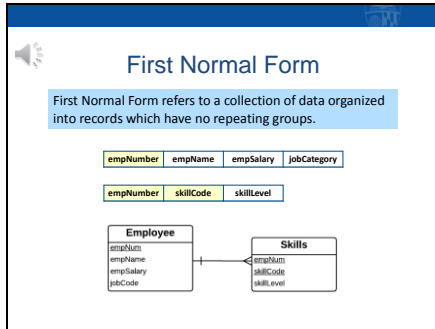
Employee name is functionally dependent on employee number because given an employee number there is a unique value of employee name. Note that the dependency is not true in the reverse direction. There can be several Jane Doe's in a database, but each would have a unique employee number.

Employee salary is also functionally dependent on employee number, as is the job category.

And... skill level is functionally dependent upon both the employee number and skill code.

Now...let's talk about normalization.

15



Data that is in first normal form has no repeating groups. The data can be stored as a simple two-dimensional table.

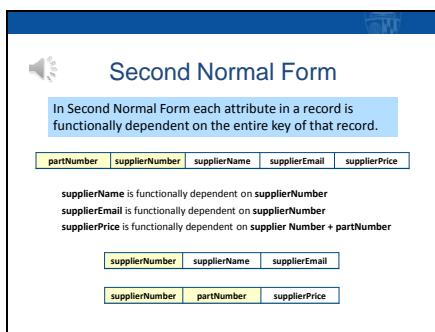
To put the data from that last example into first normal form, we'd break it up into two tables as illustrated here.

The first table will contain the employee number, employee name, employee salary and job category...and the employee number will be used as a primary key. That means it can be used to uniquely find the employee name, salary, and job category.

The second table will contain employee number, skill code, and skill level. It will contain a concatenated key consisting of employee number and skill code. We must know an employee number and a skill code to find the skill level for a specific employee.

The original data model, modified to show the normalization, looks like this . The underlined attributes indicate they are keys.

16



In second normal form, each attribute in a record is functionally dependent on the entire key of that record. When a record contains a concatenated key, there is a chance that it may not be in second normal form.

Take a look at this record. It has a concatenated key...part number and supplier number. Some of the attributes are dependent upon the entire key and some are not. Supplier name and supplier email are functionally dependent only on supplier number, but the supplier price is dependent upon part number and supplier number.

To put this data into second normal form, we split it into two tables. One table contains supplier number, supplier name, and supplier email...with supplier number as the primary key. The other contains supplier

number, part number, and supplier price...with supplier number and part number as a concatenated key.
Now...the attributes in each table are dependent upon the entire key for the table.

17

Third Normal Form

In Third Normal Form each attribute in a record is functionally dependent on the key and nothing but the key.

empNumber	empName	empSalary	projectNumber	compDate
-----------	---------	-----------	---------------	----------

empName is functionally dependent on empNumber
empSalary is functionally dependent on empNumber
projectNumber is functionally dependent on empNumber
compDate is functionally dependent on empNumber
compDate is functionally dependent on projectNumber

empNumber	empName	empSalary	projectNumber
-----------	---------	-----------	---------------

projectNumber	compDate
---------------	----------

In third normal form, each attribute in a record is functionally dependent on the entire key of that record...and nothing but the key.

Take a look at this record. Employee name, salary, project number, and project completion date are dependent on the employee number primary key . But, completion date is also functionally dependent upon project number, which is not a key. Thus...the data is not in third normal form. This situation, in which an attribute is functionally dependent on a non-key field, is called transitive dependence. Transitive dependencies must be removed in order to get the data into third normal form.

So...to put this data into third normal form, we split it into two tables. One table contains employee number, employee name, salary, and project number...with employee number as the primary key. The other table contains project number and completion date...with project number as the primary key.