Data Admin Concepts & Database Management

Table of Contents

Lab 02 – Conceptual Modeling

Overview

This lab is the second of ten labs in which we will build a database using the systematic approach covered in the asynchronous material. Each successive lab will build upon the one before and can be a useful guide for building your own database projects in a systematic way.

In this lab, we will examine narrative descriptions of organizational data needs and parse them for business rules and facts which we will model. Part 1 of this lab explores three independent case studies and asks you to model the conceptual design with an increasing level of independent work required to identify the business rules contained in the narrative. Part 2 returns to our VidCast database and provides more detail into how the organization will need to manage its data and asks you to craft the business rules and conceptual entity-relationship diagram (ERD) that implements these rules.

Read this lab document once through before beginning.

Learning Objectives

In this lab you will

- Demonstrate understanding of the conceptual model
- Demonstrate ability to parse technical requirements from non-technical narratives
- Identify the appropriate cardinality and degree of relationships
- Identify entities from a given data problem
- Identify attributes and their properties for entities in a given data problem

Lab Goals

This lab consists of two sections. The first section presents three case studies for building conceptual model ERDs. Each consecutive case study will present less and less technical information. By the end of this section, you should be able to extract technical business rules from non-technical narratives to begin the process of designing the database. In the second section, you will apply your ability to parse narratives for technical business rules and build a conceptual ERD for the recurring database process, VidCast.

What You Will Need to Begin

- This document
- An active Internet connection
- Visit draw.io (https://www.draw.io/) and ensure you can create a blank diagram. Ideally, connect draw.io to your Google Account, OneDrive, or local disk. If you are new to using online software tools, the recommendation is to connect draw.io to your Google Drive. If you do not have a drive, visit drive.google.com (https://drive.google.com) to create one for free before beginning the second part of this lab.
- A blank Word (or similar) document into which you can place your answers. Please include your name, the current date, and the lab number on this document. Please also number your responses, indicating which part and question of the lab to which the answer pertains. Word docx format is preferred. If using another word processing application, please convert the document to pdf before submitting your work to ensure your instructor can open the file.
- To have completed Lab 01 – The Relational Data Model
- To be proficient with ERD drawing tools available in Draw.io.

Part 1 – Conceptual Modeling Case Studies

Setup

The following case studies have been adapted to allow you to build and demonstrate aptitude in translating written non-technical narratives to ERDs. The first case study walks you through step-by-step instructions for identifying facts, or business rules, in the narrative to be modeled. By the third case study, you should be able to apply these principles on your own to design the solution. Each time you

complete a case study, export the diagram as an image file (PNG or similar) and insert it into your answer doc.

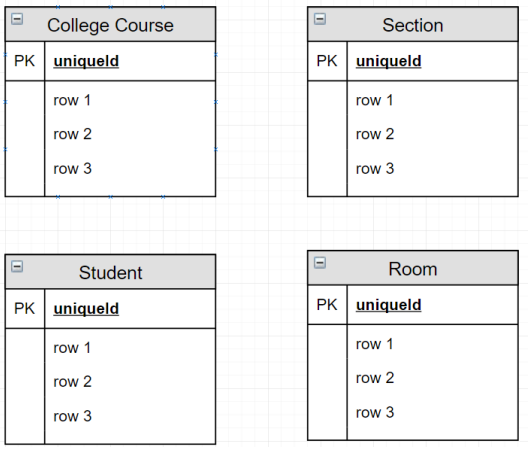
Case Study 1 – Obligatory College Classes Modeling

Consider the following narrative description of the data needs of a college.

> A college course may or may not have one or more scheduled sections. Each student may enroll in a course section, and each section can be enrolled in by many students. The capacity for a given section is determined by which room in which the section is held. Each section is only in one room, but each room can be used for many sections.
> For each course, we will need to know the course number*, name, number of credits, and a brief description. A course may also have one or more prerequisite courses, meaning a student must have completed another course before taking this one. A section should indicate meeting days and times and include the instructor's name. We will need the name, id number*, and campus email address for each student. For room, we will need the building name, room number, and capacity.
>
> ** = this identifies the entity*

We need to identify the entities, attributes, and relationships in the above paragraphs and model them in Draw.io. A good first step is to pick out the entities. Looking at the narrative, there are at least four entities: College Course, Section, Student, and Room. There may be more, but let's focus on those four.

In draw.io, add four ER Table 1 shapes to the page and rename them as College Course, Section, Student, and Room.



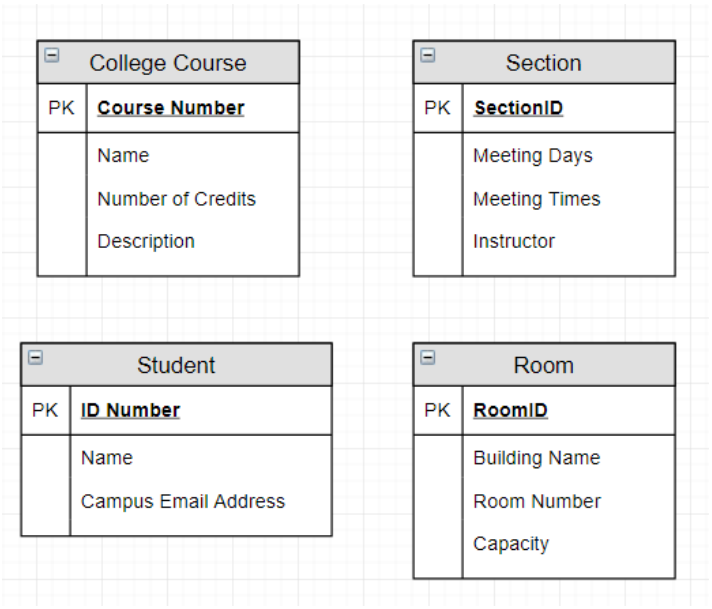
Next, let's identify the attributes described in the narrative, and put them with the entity they represent. Attributes usually describe the entity in some way. From the narrative, we see these attributes:

Entity	**Attributes**
College Course	Course Number* Name

	Number of Credits Description
Section	Meeting Days Meeting Times Instructor
Student	Name ID Number* Campus Email Address
Room	Building Name Room Number Capacity
A "*" indicates that this attribute identifies the instance of the entity. We'll put this on the PK line for each attribute.	

We need to add these attributes to our diagram. Replace the placeholder text for each entity with the attribute shown in the table. For the attributes that identity the instances of entities, place them in the "uniqueidentifier" position, as they will be the Primary Keys for those entities. Note that we don't have a good identifying attribute in the data for Section or Room, so let's make the uniqueidentifier attributes SectionID and RoomID respectively. These will be surrogate keys, or primary keys that the system will generate to help us uniquely identify instances of the entities.

Your diagram should look like this:



Note that we will not be using the third row in the student entity, so delete it by right clicking on "row 3" and click Delete from the context menu. Alternatively, click once on "row 3" to select only that object in the shape and press Delete on your keyboard. (Undo if you accidentally delete something you did not mean to delete)

You may also have to resize some of your entity shapes to fit all the text.

Next, we need to identify the relationships in this model. A relationship describes how instances of entities relate to instances of other entities. A good tactic is to look for verb phrases in the text. These are often indicative of a relationship between entities.

The relationships in the narrative above include the following:

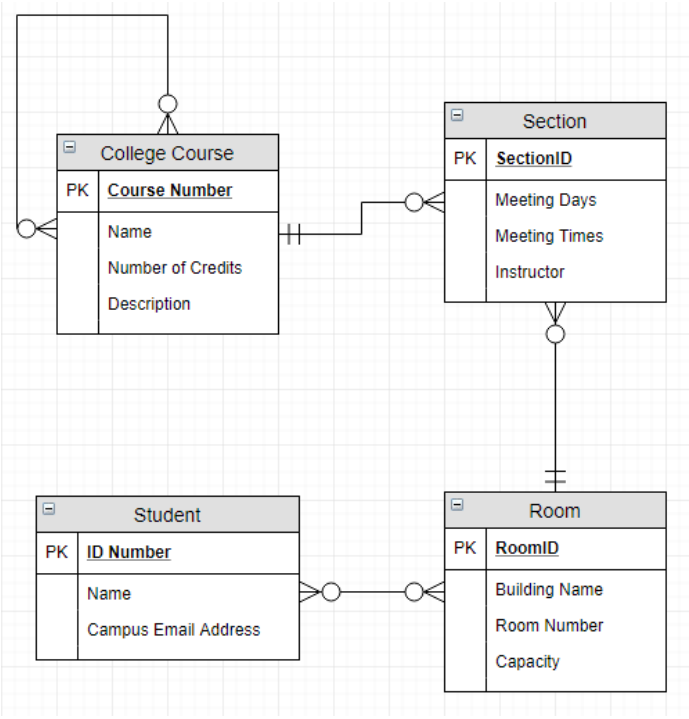
- A college course **may or may not have one or more** scheduled sections
- Each student **may enroll** in a course section, and each section **can be enrolled in by many** students
- Each section **is only in one** room, but each room **can be used for** many sections
- A course **may also have one or more** prerequisite courses

The bold emphasis in the above is a clue to the cardinality of these relationships. In draw.io, make the following connections (see the ensuing diagram for hints on which ends to use):

- One and Only One College Course to Zero or Many Sections
- Zero or Many Students to Zero or Many Sections
- Zero or Many Sections to One and Only One Room
- Zero or Many Courses to Zero or Many Courses

Note that the last one is a unary relationship, a relationship in which only one entity is involved, so the connection need to be made to itself. This can be a little difficult with draw.io, but it can be done.

Your diagram should look like the following:



Case Study 2 – Project Management

The following narrative describes a simplified database for use in an agile project management application. To those that are familiar with Agile methodology, note that this is highly simplified and

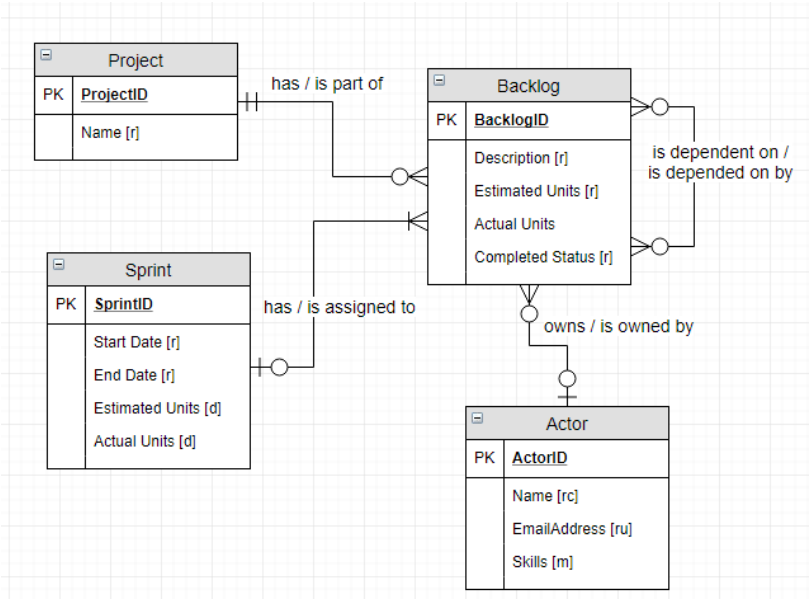
isn't intended to model the entire process. We will extend our drawing from this point forward by adding the verbiage of each relationship to the line between entities. We'll also be applying some properties of each attribute to the diagram.

> An agile project has a unique project name and is broken down into several tasks contained in a list called Backlog. Each backlog has a required description and requires an estimated number of units required for completion. Time-boxed Sprints are created to do work on backlog tasks, so each Sprint has a list of Backlog Items to be performed assigned to it, but a Backlog may not be a part of a Sprint. The start and end dates of each sprint are required. We also need to calculate the Estimated and Actual Units of Work needed and used to complete the Sprint. These are derived from the values on each Backlog assigned to the Sprint.
> Each Backlog assigned to a Sprint as an Actor assigned to it (but unassigned Backlogs do not have Actors assigned). This Actor is the person who owns the Backlog and is responsible for its completion and may not be assigned to any Backlog tasks. Each Backlog assigned to a Sprint has a completion state (yes or no) and Estimated and Actual Units taken to complete the task. A Backlog may be dependent on zero or more Backlog items, and each Backlog may have dependencies in zero or more other Backlogs. For instance, a Backlog item with a description of "install software on the server" may be dependent on another Backlog with the description, "install server and build operating system".
> For each Actor, we require their name (comprised of First name and Last Name), their email address, and a list of their skills. Because each person (Actor) should have their own email address, we want email to be required and unique.

Here's the list of entities, attributes, and relationships gleaned from the narrative:

Entity	**Attribute**
Project	Name [r]
Backlog	Description [r] Estimated Units [r] Actual Units Completion Status [r]
Sprint	Start Date [r] End Date [r] Estimated Units [d] Actual Units [d]
Actor	Name [rc] Email Address [ru] Skills [m]
Relationships	
Each Project has 1 or more Backlog, Each Backlog is part of 1 and only 1 Project Each Sprint has 1 or more Backlog, each Backlog is assigned to zero or 1 Sprint Each Actor works on zero or more Backlog, each Backlog is worked on by zero or 1 Actor Each Backlog is dependent on and is depended on by zero or more Backlog	

Draw the above in draw.io. Use surrogate keys for each entity's unique identifier. For each shape that does not have enough rows in the template, right click a row (other than the PK row) and click Duplicate. Your diagram should look like the following:



Case Study 3 – Book Publishing Database

In this example, you'll identify the entities, attributes, and relationships yourself and model them using draw.io.

> Each publisher has a unique name; a mailing address and telephone number are also kept on each publisher. A publisher publishes one or more books; a book is published by exactly one publisher. A book is identified by its ISBN, and other attributes are title, price, and number of pages. Each book is written by one or more authors; an author writes one or more books, potentially for different publishers. Each author is uniquely described by an author ID, and we know each author's name and address. Each author is paid a certain royalty rate on each book he or she authors, which potentially varies for each book and for each author. An author receives a separate royalty check for each book he or she writes. Each check is identified by its check number, and we also keep track of the date and amount of each check.

Part 2 – VidCast Conceptual Model

Setup

In the previous lab, we looked at some sample data collected for our video casting service. After reviewing the data and having more conversations with the product manager, we have determined the following facts about our data. Use these facts to construct an ERD in draw.io.

Each user is identified by a User ID and has a unique user name and a unique email address, both of which are required. A user can optionally include a description of themselves and a URL to their own web sites. Each user can have many other users as followers and can, in turn, follow many other users

(think unary relationship!). To help users find one another, each user can add categorizing tags to their profile. Each of these tags can be applied to many other users as well.

Vidcasts are broadcasts of a video stream, identified by a system-generated VidCast ID. A vidcast must have a title. A vidcast may have a start date and time and a projected duration. This duration is replaced with the actual duration. A Vidcast can also be recorded. If it is, the recording will be stored on a secure web service such as Amazon Web Services S3. We will need to log the URL of this recoding with the vidcast. Each vidcast is made by exactly one user, but each user can optionally create many vidcasts. The user can tag each vidcast using the same tag list as is used for user tagging. Because vidcasts can be scheduled ahead of time, each vidcast requires a status (Scheduled, Started, Stopped, Cancelled are examples of these statuses).

To-do

For the above narrative, use draw.io to create a conceptual ERD for the VidCast software. Once finished, export it as an image and place it at the end of your answer doc.

After completing Part 2, save and submit your answer doc.