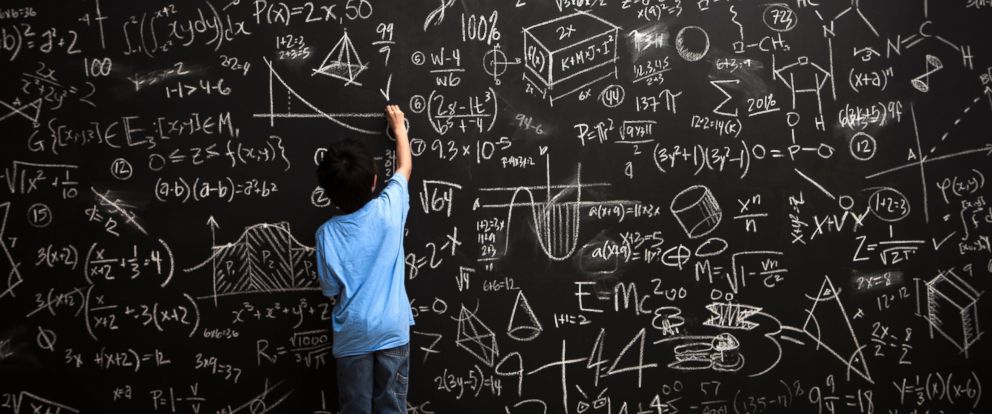
CSC370 - Assignment 2 - Daniel Dubichev

Relational Algebra, E/R to Tables, Key Mapping & Normalization.



Daniel Dubichev – V00877776 – CSC370 – Assignment/Project Phase two.

***Introduction***

This document explains and shows the process of converting an E/R diagram into a relational database schema. The reasoning for this phase is to have an idea of what our SQL tables will look like, as well as how they are mapped to each other. To begin, I will provide a review of my previous E/R Diagram, highlighting some important components and my methodology to translate into relational schema/table form. As well, I will segment this process into five parts, which are:

schema definitions,

normalization and E/R to relational table translations,

mapping of keys and column attributes/domains and their constraints for tables,

relational algebra/sql queries to be performed on database,

conclusion

Normalization will be my biggest asset in the design of the relational schema, because a good looking and functioning database (in terms of time and space, reducing row redundancy) should be in 3NF form. I will stress this throughout this document, as from my short review below, translating the data should be fairly simple, as it is just a relational representation of our E/R diagram, but the normalization will require some effort.

***Review of components***

Drawn in **Figure 1** is my E/R Diagram from Phase 1 of the project. There are three main entities that compromise the entire Diagram, being **User, Material and Sequence.** After all, **User’s** select **Material** and form a **Sequence** for other users to access. In terms of directly translating the E/R diagram into a Relational Schema, this poses no major difficulty, as we are just are defining the schemas as:

schemaName( attribute1 , … , attribute n )

Where the ‘**schemaName**’ is the specific entity name in the E/R Diagram such as **User, Material, Sequence**. Further entities such as **Entry**, or the subclasses of **User, Staff** and **Learner,** will be explained later (normalization) but we will just focus on the big three for now **(User, Material and Sequence).** These schemaName’s are represented by Rectangles on the E/R Diagram.

**‘( attribute1 , … , attribute n )’** compromises of the attributes (or in visual terms, columns) for that entity for which it is connected by a line. For instance, the **attributes of User** would be

userName

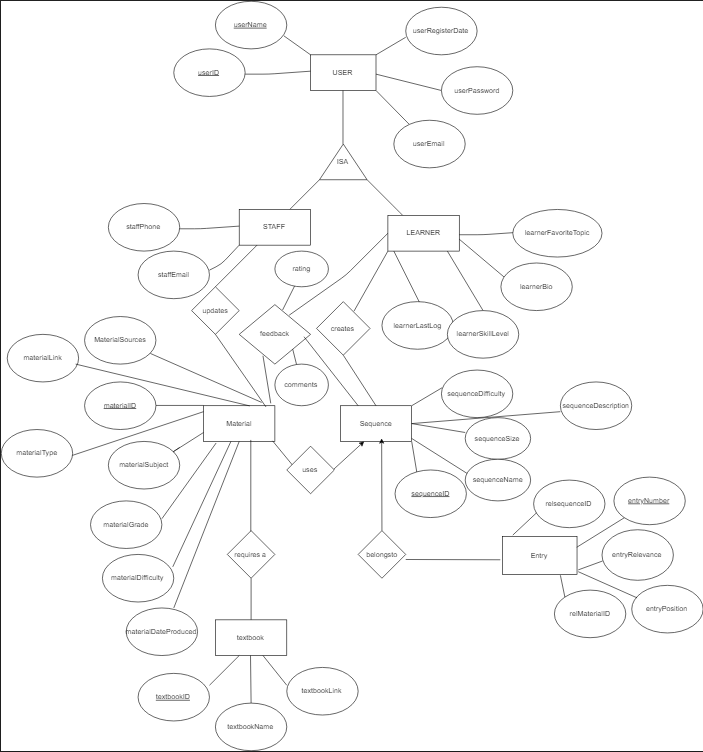
userID

userRegisterDate

userPassword

userEmail.

Knowing the two components of the relational schema, schemaName and attributes, we may apply them together to create the basic relational schemas. This process is found after the **Figure 1** visual.



**Figure 1: Final E/R Diagram for Phase 1.**

***Relational Schemas – before the normalization, constraints and key mapping***

Before the implementation of normalization, I believe it would be a good strategy to translate each schema, compromising of an entity and it’s attributes directly from the E/R Diagram. After the translation, we should be able to formalize and refine each schema’s normalization, as well as define the primary, foreign keys and domains.

**User**

The user entity has two subclasses to account for while also considering the entity itself, So a user may have **Learner** or **Staff** privileges.

**USER**(userID, userName, userRegisterDate, userPassword, userEmail)

**LEARNER**(learnerFavoriteTopic, learnerBio, learnerSkillLevel, learnerLastlog)

**STAFF**(staffPhone, staffEmail)

The Learners, are users who create sequences and are able to view/browse Material in order to create said sequences. The Staff, have a relationship with Material as they are able to browse and update the Material if a learner requests a new material to add to the collection, or if they simply want to

**Material**

The material entity consists of multiple attributes corresponding to what kind of material it is, what subject it relates to and more as defined to the attributes posted in **Figure 1.** It also has relationships with Learner, Sequences and Staff

**MATERIAL**(materialID, materialSource, materialLink, materialType, materialSubject, materialGrade, materialDifficulty, materialDateProduced, textbookID)

We haven’t entered the Normalization section, but I implemented some in the E/R Diagram by having a separate **Textbook** entity, this is because not every Material will require a textbook, and some may need the same textbook, so instead of having a long sequence of characters, we may simply refer to the textbook using **textbookID.**

**TEXTBOOK**(textbookID, textbookName, textbookLink)

**Sequence**

Sequence is an interesting entity to say the least. A sequence for this project is a series of chosen materials, like a courses progression in school, it would have multiple units. Further, sequences have heavily implied normalization from the fact that a sequence contains multiple entries. So, there will be another entity titled **Entry**, which relates each entry to a sequence. For example, Imagine a sequence with one thousand entries. For each entry, we would need have an instance of that sequence with all of it’s attributes including the information for that entire sequence a thousand times. Having the **Entry** entity references the sequence ID for each entry, so instead of having a thousand entries in the sequence table, we can only have one entry and reference the entries using a relationship we will explore later in this document (key section). Learner users can create and browse sequences, while sequences are compromised of resources from the material section

**SEQUENCE**(sequenceDifficulty, sequenceDescription, sequenceSize, sequenceName, sequenceID)

**ENTRY**(relsequenceID, relmaterialID, entryNumber, entryRelevance, entryPosition, entryPosition)

Thus concludes the process of converting each entity and their respective attributes from the E/R diagram into their respective schema formats. However, there are still some refinements to be made before drawing the tables for each schema. For example, the User subclasses, Learner and Staff, have attributes unique to their roles in the company, but there is no relUserID to relate that actual information to the User entity itself, which compromises of attributes that both a Learner and Staff member would share!

***Schema Refinement - Normalization***

There exists a few schemas that could use a bit of refinement before drawing tables for them. Starting with the schema **User** and its subclasses, Learner and Staff. Learner and staff have unique attributes specific to the roles in their company such as a Staff having their phone number, for quick work related contact, unlike a Learner who doesn’t need their phone number. However they do share a common attributes, like a username or password. So in order for a learner/staff’s attributes to be connected with it’s shared User attribute, it should have a relUserID attribute.

**USER**(userID, userName, userRegisterDate, userPassword, userEmail)

**LEARNER**(learnerFavoriteTopic, learnerBio, learnerSkillLevel, learnerLastLog, reluserID)

**STAFF**(staffPhone, staffEmail, relUserID)

For **Materials** I couldn’t find many attributes that would be necessary to refine/normalize, let’s look over all of the attributes to explain what I mean:

**materialID**: This will be defined as the key later on. Therefore there is no real performance saving reason to refine.

**materialSource**: Every Material will have a source, indicating where it came from. Without a proper source, the material may be considered unprofessional and not suitable for the degree of quality for our database. A source would be a University, Professor/Author, or even a Publication company. There is no further information we need to associate with the source rather than the source itself, therefore normalization wont be necessary.

**materialLink**: A link to a pdf/online document/video/file etc… of this specific singular material item. More likely to be a URL. No further information about the link is needed, so normalization isn’t needed.

**materialType**: What type of material is it? A pdf / scholarly article / powerpoint presentation? This field could be normalized to it’s own table, but information about what type of article it is isn’t very mandatory for our database.

**materialSubject**: What is the subject of this certain material? Not very mandatory to split this in its own table. The material itself will surely provide all sorts of information relevant to the Subject

**materialGrade**: What grade is the material intended for. From kindergarten to graduate school. Again no information is going to be needed about the grade, so it doesn’t deserve it’s own table

**materialDifficulty** & **materialDateProduced** also don’t require normalization for similar reasons as the other attributes explained above.

**Textbook** however is a separate story. The key takeaway with textbook is that they can be up to 100 characters or longer for example:

Davis’s Guide for Nurses (15th edition) by April Vallerand and Cynthia A. Sanoski ISBN: 9780803657052

Say there are 100 materials with that textbook, this would take a lot of space. An alternative is to normalize textbook’s by giving them an ID, and having Material table reference an ID instead of a 100 character string for each textbook entry.

As explained in project specifications, it also noteworthy to mention that no material values may be null except for textbook ID, as the material may not require a textbook explicitly (4 hour long youtube tutorial on grade 2 algebra for example).

My methodology for my process to only normalize can be best seen in **Figure 2**.



**Figure 2. A basic row in the Material referencing a homework and its attributes. Notice that textbook is null. No other values are allowed to be null however. Textbook is normalized, but the other attributes are not because each piece of material is unique.**

It may improve performance to convert and relate each attribute to it’s own table, as we did with Textbook, but for the sake of simplicity, and the **important guarantee of non duplicate material imposed by the company staff**, data duplication shouldn’t be a worry.

For **Sequence** & **Entry,** there are no specific refinements/normalizations we need to implement. The paradigm of separating entries from the sequence is one of the most important of this project, and ultimately will save massive amounts of space.

Starting with **Sequence**, I will go through each attribute and explain why the normalization applied is already excellent from a design perspective.

Attributes such as **sequenceDescription** and **sequenceName** are such arbitrary attributes that it would be counterproductive to normalize tables for them, as it is very improbable that someone would choose the exact same name for a sequence of materials, let alone the exact same description of a sequence. The probability of duplicate data in this case is extremely low.

More, the **sequenceSize** is going to be an integer, so normalizing this row value would be beyond redundant. This also applies to **sequenceDifficulty**, which is a number on a scale of one to ten.

**sequenceID** will be the primary key, so normalizing this makes no sense.

Now Sequence is properly normalized, it is time to look at **Entry** and it’s normalization properties.

Each Entry belongs to a sequenceID. This is established via **relSequenceID**. This is because of reasons explained earlier, that separating entries from the actual sequence table would save immense amount of time, as sequences may contain dozens, perhaps even hundreds of entries. Each Entry has information as well, such as the position of the entry in that specific sequence, announced by **entryPosition. entryRelevance** is a descriptive attribute that entails the reason why that entry belongs to the related sequence, as well why it is in that specific position. **refMaterialID** references the article of material that the entry corresponds to. **entryNumber** is the primary key of the entry.

To conclude the normalization process, there were a few adjustments as seen as above, but overall, the entity/relationship diagram translated quite nicely into normalized relational schemas. I think that giving each material attribute it’s own table would be viable for performance, but it violates the simplicity constraint that is often so emphasized in database design. As well, the staff of the online database would be able to access and delete any duplicate entries of material. Now, onto the constraints and key mapping.

**Key, Constraints and Domain Mapping**

I will now map the constraints for each relational schema, giving more of a formal, structured feel to the tables now that they have been extensively normalized and refined. I will go over each attribute/column for the corresponding schema.

**User**

userID – int NOT NULL AUTO\_INCREMENT

PRIMARY KEY (Userid)

will be the ID used to reference any instance of a user. Cannot be null. Also auto increments every time a new value is added to the DB

userName – varchar(21) NOT NULL

I chose varchar because it will save us padding/whitespace if the UserName is under 21 characters. Also, 21 characters is a fine username length, giving people some lengthy options for the sacrifice of data storage will be a nice compliment with users. Shouldn’t be null.

userEmail – varchar(75) NOT NULL

75 is a fair assumption for an email length. This attribute is just to store emails, so it shouldn’t be null

userPassword – varchar(21, 6) NOT NULL

The password will not use a password hashing algorithm, they are fundamentally sound, but will not be considered in this project for simplicity. However, passwords must be 6 character of length. Also, they may not be null.

userRegisterDate – Date

Used for information about the user. Should be combined with a function to immediately receive date after the registration is made.

**Learner**

learnerID int,

FOREIGN KEY (learnerID) REFERENCES User(userID)

learnerFavoriteTopic varchar(20)

Doesn’t need to be null. Used to match with familiar sequences

learnerBio varchar(80)

A nice short bio summarizing the users other’s may learn about one another. Can be null

learnerSkillLevel varchar(15)

Easy, Medium, Advanced, Beginner, Professional, Admin are just some examples. 15 characters or less is suitable. Used to find familiar sequences along skill level, but can be null

learnerLastLog date NOT NULL

Used for information about the user. Should be combined with a function to immediately receive date after initial log is made.

**Staff**

StaffID int

FOREIGN KEY (StaffID) REFERENCES User(UserID)

StaffPhone varchar(15) NOT NULL

Staff will have phone numbers to call in case of emergency. Also for general communication. Not null.

StaffEmail varchar(75) NOT NULL

75 is a fair assumption for an email length. This attribute is just to store emails, so it shouldn’t be null.

**Materials**

No null values permitted except textbookID, which may be a secondary resource to a material. No nulls are implemented to support users, to minimize query restrictions and to have many querying options, such as date of material produced, or material source (Uvic for example) to find their perfect educational material. Only full, well rounded queries permitted.

**materialID**: int NOT NULL PRIMARY KEY AUTO\_INCREMENT

Primary key for material. Not null and auto incrementing

**materialSource**: varchar(30) NOT NULL

Almost like a citation, who is the creator of this source. Cannot be null as users may query the materials by source

**materialLink**: varchar(100) Not null

A link to a pdf/online document/video/file etc… of this specific singular material item. More likely to be a URL. Cannot be null because the very fundamental of our database is that it carries links to scholarly material across the internet

**materialType**: varchar(20) NOT NULL

What type of material is it? A pdf / scholarly article / powerpoint presentation? Cannot be null for queries by users

**materialSubject**: varchar(30) NOT NULL

What is the subject of this certain material? Cannot be a null value for user querying

**materialGrade**: varchar(15) NOT NULL

What grade is the material intended for. From kindergarten to graduate school. Cannot be a null value for user querying

**materialDifficulty**: smallint NOT NULL

Scale of one to ten on how difficult the material is. Cannot be a null value for user querying. Smallint for performance.

**materialDateProduced** date NOT NULL

Date value that is also not null, for users may query the produced

**textBookID:** int

integer for each textbook. There is a foreign key that references this in the textbook relation. Can be null if no textbook is required

**Textbook** – result of normalization in the relational process. For one textbook id in Material there will be a row in textbook.

**reltextbookID int**

FOREIGN KEY (reltextbookID) REFERENCES material(textbookID)

References a textbookID in material table, each material’s textbook, if they have one, will have textbook attributes associated to the corresponding related textbook.

**textbookLink varchar(100) NOT NULL**

URL link to buy a textbook or view one online. May not be null

**textbookName varchar(75) NOT NULL**

textbooks may have long names, so I feel varchar(75) is fair. Cannot have a null value.

**Sequence**

**sequenceID int,**

**PRIMARY KEY(sequenceID)**

Primary key for each sequence

**sequenceDescription varchar(100)**

The description is a short sequence of characters used to inform other users about what the sequence consists of. This is optional and therefore may be null

**sequenceSize smallint NOT NULL**

How many materials are in the sequence? Not null for ease of querying

**sequenceName varchar(40) NOT NULL**

Name of sequence can’t be null because of the querying by name of sequence methodology

**sequenceDifficulty smallint NOT NULL**

Difficulty of sequence on a scale of one to ten. Not null for easy querying.

**Entry**

**relsequenceID int NOT NULL**

**FOREIGN KEY (relsequenceID) references sequence(sequenceID)**

**relmaterialID int NOT NULL**

**FOREIGN KEY (relmaterialID) references material(materialID)**

These two keys identify what the qualities and characteristics of the entry from the material table, by knowing the materialID we may head to the material ID and find all the attributes associated with the unique material (ID) to discover more about it. The sequence ID corresponds that entry of material to a certain sequence

**entryRelevance varchar(50)**

Quick sentence about why this specific entry is relevant to the sequence, or why it’s in that position. Can be null

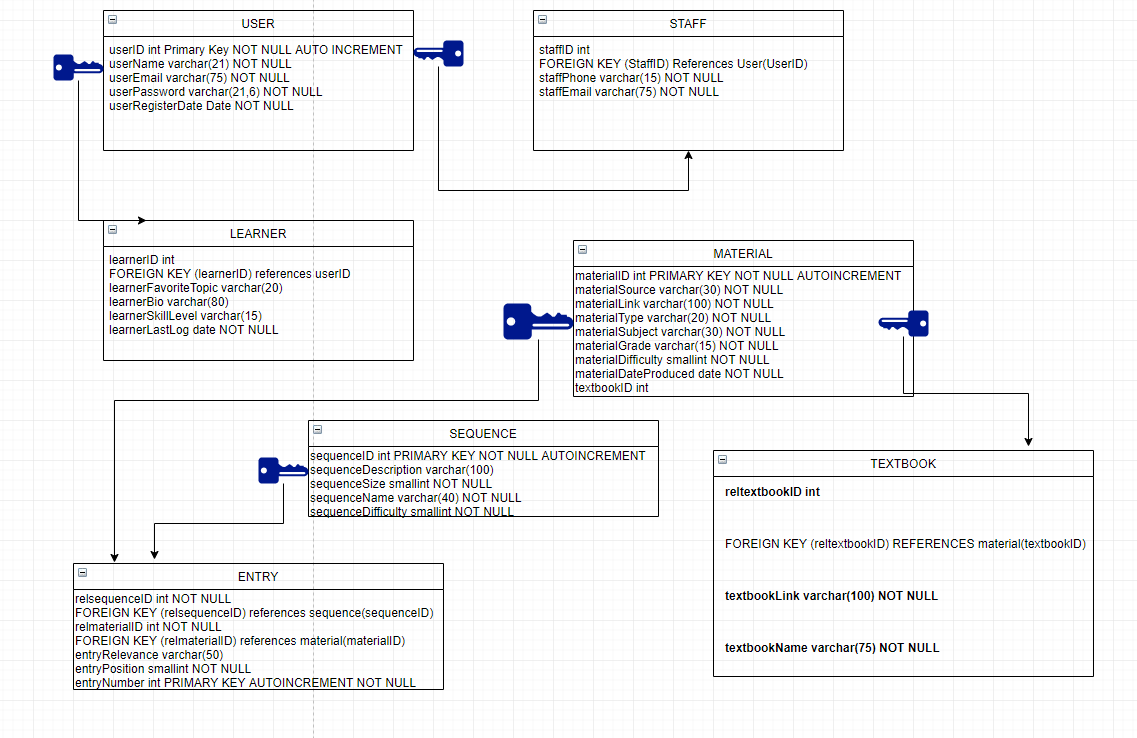
**entryPosition smallint NOT NULL**

integer to denote which position the specific entry is in for that sequence denoted by relsequenceID. Not null, must maintain order of positions of entries

**entryNumber int PRIMARY KEY AUTO INCREMENT NOT NULL**

Primary key for all entries, to distinguish each one independently

Visual representation of relational schemas with fully applied normalization, constraints, domains and keys.



The blue key and arrow denote relationship roles. Blue key would be the primary key, and the arrow represents the relation in which the primary key is derived to form a foreign key.

***Some relational algebra and queries*** π, δ

1. Select all materials that are from the University of Victoria
2. Select all materials that are youtube videos and from 2003
3. Select only the type of materials that have a difficulty of 2/10
4. Display the learner ID of all learners who’s favorite topic is relational algebra
5. δ(materialSource = ‘University of Victoria’)(Material)
6. δ(materialType = ‘Youtube’ AND year(materialDateProduced )= ‘ 2003’)(Material)
7. π(materialType) ((δ(materialDifficulty = ‘2’)(Material))
8. π(learnerID)(( δ(learnerFavoriteTopic = ‘Relational Algebra’)(Learner))

***Conclusion***

In Conclusion, We have successfully converted our E/R Diagram into a relational schema. The most demanding aspect of this process was by far the repetitiveness of defining domains for columns/attributes. There were some significant normalization paradigms implemented in the E/R diagram, which simplified many things for me, however I still decided to not normalize everything to the fullest extent for the sake of keeping it simple. I don’t have a journal for this assignment, but I most definitely documented everything in file. I am excited to begin Phase 3 of the assignment, and ready to start hands-on coding my project.