|  |
| --- |
|  |
| Preparation and Participation (Blackboard Discussions) |
| Course Section: CS605.641.81  Summer, 2020 |
| Prepared by |
| **Brian Loughran** |
| **08/06/2020** |

|  |
| --- |
|  |

**Table of Contents**

[Summary 4](#_Toc516817920)

[1. Original Posts 4](#_Toc516817921)

[1.1 Discussion 1A 4](#_Toc516817922)

[1.2 Discussion 1B 5](#_Toc516817923)

[1.3 Discussion 1C 5](#_Toc516817924)

[1.4 Discussion 2A 7](#_Toc516817925)

[1.5 Discussion 2B 7](#_Toc516817926)

[1.6 Discussion 2C 8](#_Toc516817927)

[1.7 Discussion 3A 8](#_Toc516817928)

[1.8 Discussion 3B 9](#_Toc516817929)

[1.9 Discussion 3C 10](#_Toc516817930)

[1.10 Discussion 4A 10](#_Toc516817931)

[1.11 Discussion 4B 12](#_Toc516817932)

[1.12 Discussion 5A 13](#_Toc516817933)

[1.13 Discussion 5B 14](#_Toc516817934)

[1.14 Discussion 6A 15](#_Toc516817935)

[1.15 Discussion 7A 16](#_Toc516817936)

[1.16 Discussion 7B 17](#_Toc516817937)

[1.17 Discussion 8A 22](#_Toc516817938)

[1.18 Discussion 8B 24](#_Toc516817939)

[1.19 Discussion 9A 25](#_Toc516817940)

[1.20 Discussion 9B 26](#_Toc516817941)

[1.21 Discussion 10A 27](#_Toc516817942)

[1.22 Discussion 10B 28](#_Toc516817943)

[1.23 Discussion 11A 29](#_Toc516817944)

[1.24 Discussion 11B 29](#_Toc516817945)

[2. Replies to Classmates' Posts 30](#_Toc516817946)

[2.1 Discussion 1A 30](#_Toc516817947)

[2.2 Discussion 1B 31](#_Toc516817948)

[2.3 Discussion 1C 31](#_Toc516817949)

[2.4 Discussion 2A 31](#_Toc516817950)

[2.5 Discussion 2B 32](#_Toc516817951)

[2.6 Discussion 2C 32](#_Toc516817952)

[2.7 Discussion 3A 33](#_Toc516817953)

[2.8 Discussion 3B 33](#_Toc516817954)

[2.9 Discussion 3C 34](#_Toc516817955)

[2.10 Discussion 4A 34](#_Toc516817956)

[2.11 Discussion 4B 35](#_Toc516817957)

[2.12 Discussion 5A 35](#_Toc516817958)

[2.13 Discussion 5B 36](#_Toc516817959)

[2.14 Discussion 6A 36](#_Toc516817960)

[2.15 Discussion 7A 37](#_Toc516817961)

[2.16 Discussion 7B 37](#_Toc516817962)

[2.17 Discussion 8A 37](#_Toc516817963)

[2.18 Discussion 8B 38](#_Toc516817964)

[2.19 Discussion 9A 38](#_Toc516817965)

[2.20 Discussion 9B 39](#_Toc516817966)

[2.21 Discussion 10A 39](#_Toc516817967)

[2.22 Discussion 10B 40](#_Toc516817968)

[2.23 Discussion 11A 40](#_Toc516817969)

[2.24 Discussion 11B 41](#_Toc516817970)

**Preparation and Participation (Blackboard Discussions)**

SUMMARY

I have attached 60 discussion forum participation examples that I made throughout the semester. The first 24 are examples of my original posts. The final 36 are examples of my responses to other students’ posts.

|  |  |  |
| --- | --- | --- |
| **Module** | **Total Original Posts** | **Replies to Classmates' Posts** |
| Module 1 | 3 | 4 |
| Module 2 | 3 | 4 |
| Module 3 | 3 | 5 |
| Module 4 | 2 | 4 |
| Module 5 | 2 | 3 |
| Module 6 | 1 | 2 |
| Module 7 | 2 | 2 |
| Module 8 | 2 | 3 |
| Module 9 | 2 | 3 |
| Module 10 | 2 | 4 |
| Module 11 | 2 | 2 |
| All Modules | 24 | 36 |

# ORIGINAL POSTS

## Discussion 1A

May 26, 2020 6:53:06 PM EDT

Hello class, I am Brian Loughran, and I will be taking this course over the duration of the summer with the rest of you. My educational background is in Mechanical Engineering, I graduated Lehigh in 2017 with a B.S. and went on to work at Pratt and Whitney, an aerospace company near Hartford, CT. At Pratt, I have worked in numerous roles as part of a rotation program, including as a structural engineer for operational commercial engines, and as an aerodynamic engineer for emerging commercial fan programs. My current role is as a software developer, working on creating an automated inspection/repair service for airfoils damaged in the field. I also have experience in mobile app development, however this is more of a hobby at this point.

The reason I am taking this class is that I realize I have no formal knowledge of database systems or database management, everything I know about databases I learned either from stackoverflow or informal learning on the job in my current role. I realized a few months ago this was a weakness of mine, so I decided to take this class to try to turn database knowledge into more of a strength. Hopefully by the end of this class I will be more comfortable with database systems and their implementation and will be able to use that knowledge in my current role.

As for what I like to do in my free time, I try my best to be active. I play in a hockey league all year round as well as a soccer league during the warmer half of the year. During the colder half of the year I like to go skiing as much as possible, whether that be in the Northeast or the occasional trip out west. Since none of those activities are permitted per covid, recently I have taken up running (which I enjoy much less, but I have had to be more creative in finding activities that allow me to keep my 6 feet) and golf. There is a photo of me below from my western ski trip this year to Big Sky in Montana (pre covid):

[photo](https://blackboard.jhu.edu/courses/1/EN.605.641.81.SU20/db/_9814712_1/embedded/IMG_0324.JPG)

## Discussion 1B

May 27, 2020 6:37:59 PM EDT

I will admit that I am a bit of a noob in terms of database design/modeling knowledge. I have no formal database training (like a course or certificate relating to databases), nor do I have any experience in databases from my work experience. The only database experience that I do have is in working on an iOS app in my free time. The database that I used for the app is in Firebase, which is a cloud hosted, NoSQL database to store user information as well as some public data for the users to interact with.

Since I have very little knowledge on databases, I am excited to learn as much as I can through this course. i am sure that throughout my professional career I will have to be able to design and implement a variety of databases (I am currently developing applications in AWS at work, so I anticipate some integration with Aurora/Dynamo/etc.). I am also curious to see how the database that I set up for my iOS app matches up with the principals taught in this class, as well as any type of security practices that are common for standard databases (as most of the knowledge for that database was found on stackoverflow).

## Discussion 1C

May 29, 2020 12:03:27 PM EDT

The three schema architecture includes three levels to separate the user application and the physical database. It vaguely reminds me of the way that MVC (model view controller) architecture separates the view from the data model in classical GUI development (although the controller from MVC is not particularly similar to the conceptual level of the three schema architecture). The three schema architecture is made up of the following levels:

- External Level: The external level includes the views that the user will interact with, also know as a view schema. Each of the views on the external level displays a specific part of the database, and hides the rest of the database from view.   
- Conceptual Level: The conceptual level, also known as the conceptual schema, describes the database at the conceptual level, also known as the logic level. The conceptual level is where the logic of what data will be stored in the database as well as the relationships are between different pieces of that data.   
Internal Level: The internal level of the database describes how data is physically stored in the database, and is also known as the physical schema. Commonly lol-level data structures are described in detail in the internal level of the database.

Now that we know the different levels of the database, it becomes obvious why we need the mappings between different schema levels. Without the mapping between the external level and conceptual level, known as external/conceptual mapping, as well as the mapping between the conceptual level and the internal level, known as the conceptual/internal mapping, the transforms between the different levels could be confused. If data is not properly mapped from user input at the external level all the way down to the actual storage of the data at the internal level and vice-versa, the data stored and displayed in the database could become compromised.

Different languages are used for data definition, manipulation, query and control. All four types of database languages are crucial to having full control over the data as well as how they support the database schema. Below is a summary of each of the 4 main database languages.

- Data Definition Language (DDL): DDL is a language that allows a database administer to create a database after it has been designed. DDL commands commonly include things like CREATE, ALTER, DROP, and RENAME.  
- Data Manipulation Langauge (DML): DML is a language used to perform basic data operations on the database such as data retrieval, insert, update, delete, commit, or rollback.   
- Data Query Language (DQL): DQL is a language that supports only data retrieval. To create reports, users may have to use more complex queries using DQL.  
- Data Control Language (DCL): DCL gives user access to the database. Basic commands include GRANT, REVOKE, AUDIT, and LOCK.

Database transactions should conform to the following properties:

- Atomicity: The transaction has to work. Obviously if any part of the transaction does not work, it can result in corrupted data  
- Consistency: All transactions will navigate from one valid state to another valid state. If the database is not in a valid state at any time, you can see downtime or corrupted data.  
- Isolation: Sequential transactions happen independently with no effect on the previous or future transactions. If transactions are not isolated you can see corruption in the data.  
- Durability: Completed transactions remain even in the event of system/operational failure. If a database is not durable, you can experience data loss.

The way to remember these properties is by the acronym ACID, which stands for Atomicity, Consistency, Isolation, Durability. These characteristics are important, because if any one of these characteristics are not met in your database project you can experience data corruption or data loss.

## Discussion 2A

June 2, 2020 7:21:48 PM EDT

There are some components of the ERD that are easier to design, and some components that are harder to design. For example, one component which I generally find easy to design is each of the entities. For example, each dependant has a name, sex, birthday and relationship (to the employee). Each department has a name, number, and list of locations. Each project as a name, a number, and a location. While there are some intricacies in that some attributes may composite, some attributes may be multivalue, some attributes may not be mandatory, etc. I typically find it relatively easy to lay out the different applicable attributes of a given entity.

A component of the ERD that I find more challenging is the relationships between entities. Especially when entities have multiple 1:many relationships, like the relationship between the employee , department and project. Intricacies include the employee can either work for or manage a department, each department can have multiple or zero employees, each department can have multiple or zero projects, and each employee can work on one or many projects. It can become even more complex if you consider that each project can have multiple departments contributing (not included in the diagram, but is possible in real life). As the number of relationships increase, the complexity only grows. Managing these relationships, especially cyclical relationships as described, I find to be challenging in the design of the database. Managing identifying/non-identifying relationships and the cardinality of relationships becomes crucial in this component of database design.

Perhaps a strategy that I may employ when designing my own database is to define all the entities first. Since that part is easier for me, once the entities are all laid out, basically half of the ERD is already complete. This could help with bounding the amount of remaining complexity when I try to tackle the relationships.

## Discussion 2B

June 2, 2020 7:36:53 PM EDT

I think I am beginning to understand the ERD including the entities and relationships that are part of the diagram. While mapping out the relationships may be a bit more challenging for me than mapping out the entities, I believe I could create an ERD for another database system. One question I had concerning the ERD was if there was any software package that was recommended to create the with Chen's Notation.  I typically use drawio for most of my diagrams, but I am not sure a diagram created in drawio would look particularly similar to Figure 3.2, as drawio typically creates very blocky diagrams. Drawio might me better suited for IE notation. Any suggestions for ERD software packages would be greatly appreciated.

Going a little deeper into Chen's Notation vs. IE notation, I was curious which was more widely used, as well as if there was any preference which for which notation to use for the database design project?

I also noticed that only the waterfall software lifecycle model was described in detail in this section. Is the waterfall lifecycle model the preferred lifecycle model for database systems (rather than agile, spiral, etc.)? Or was this simply to cap the amount of information contained in the module?

As I have no database experience at work, I have not had any issues/challenges with the conceptual database design at work personally. I imagine that agreeing on entity attributes or relationships between entities might be a challenge if there are many stakeholders in a particular database system.

## Discussion 2C

June 2, 2020 7:40:39 PM EDT

The only experience in database design is with a Firebase (NoSQL) database that I created as part of a mobile app that I put together in my free time. The only design tool I used was the Firebase GUI where I could test storing and retrieving data, which is not a strategy that I would recccomend for doing the homework assignments and projects.

I an interested to see what other database design tools such as the ones mentioned in the discussion topic are suggested, as I will likely base my choice of design tool heavily on what people have found to be successful (as well as some research on my end).

## Discussion 3A

June 10, 2020 7:49:32 PM EDT

One potential design issue that I see with Chen's notation is the relative clutter of entities and their attributes and relationships. The attributes of an entity are scattered all around the entity, requiring a careful eye to identify all of the attributes of the entity. This is in contrast to IE notation, where all of the entity attributes are in a box for the entity, making them very easy to identify. Another potential issue is with the more complicated relationships between entities. For complicated entity relationships like between employee and department it can be confusing to follow if you do not understand the different relationship notations. This is similar with IE notation, however, in my opinion. as relationships between entities are difficult to represent graphically.

In converting from Chen's notation to IE, there were a few challenges for the ERD conversion. The first issue was in migrating relationship attributes between two entities. For example, between employee and department, it was challenging to figure out where the start date should go, or even what it was. If the start date is for the employee in the department, both the department and employee would have multiple start dates (if the employee has had multiple roles in the company), making it difficult to map which start date matches with what employee and what department.

Another challenge with converting Chen's notation to IE is in identifying the primary key and foreign keys for each entity. In terms of primary keys, the primary key does not seem to be a part of Chen's notation, thus a logical choice must be made for the primary key. For the foreign keys, it was sometimes difficult to determine which entity in the relationship the key should be mapped to. Generally the attribute is mapped to the child entity, however that was something of a challenge for me.

## Discussion 3B

June 10, 2020 8:41:39 PM EDT

This is good point about some of the requirements for a database project being unclear. This can be due to either incomplete requirements or due to the customer not knowing exactly what they want. In either case, it is important to have a dialog with the customer to make sure the requirements of the project are clear in the design phase, as making changes in later phases of the project can be much more expensive. For this particular database design exercise, some questions you could ask are as follows:

* Are there any upper limits on the number of courses in which a student can enroll, or the number a professor can teach?
* Is there a way to model the case where a faculty member may also be a student?
* Should each faculty member have an attribute that tracks which classes they are capable of teaching?
* Should each class have associated prerequisites?
* Should each student/faculty track what courses they have taken/taught previously?
* How should new course requests be tracked and assigned?
* Are students assigned faculty advisors for this program - is there any relationship between student and faculty?

Some reasonable assumptions are as follows:

* Relationship between students and courses should be M to N
* Enrollment should be used to track the M to N relationship between student and courses
* Relationships between faculty and courses should be 1 to M
* Each student/faculty/course should be assigned an identifying number
* Since attributes like phone number may not be provided, they should be marked optional

This should give a pretty good start between the customer and the people working the project in nailing down some of the unwritten parts of the requirements. Of course, the designers and the customer should keep in contact to ensure that the project is coming along as expected and that the designers are able to meet all project requirements.

## Discussion 3C

June 10, 2020 8:03:44 PM EDT

I have not used, nor am I familiar with any design tool used to easily draw an ERD for Chen's notation. I suppose in a bind I could use powerpoint to create the diagram, however that would likely be tedious and painful. For IE notation, MySQL workbench and draw.io are tools that I am familiar with which are capable of creating the diagram easily (although I suppose you could also do powerpoint, but that has similar issues with being tedious and painful as Chen's notation). I do not have any personal experience on database design with either notation, as I am a rookie when it comes to databases.

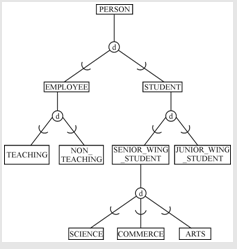
In terms of whether I prefer Chen's or IE notation, I prefer IE notation. This is due to the relative cleanliness and simplicity of IE notation in relation to Chen's notation. Chen's notation tends to have attributes all over the place, relationships are represented by complicated figures, and it feels harder to absorb for me. IE notation is relatively clean and compact in comparison. Using database design tools increase productivity in reducing necessary communication between stakeholders in the organization. A well laid out ERD will clearly convey to all stakeholders what the goals of the database are, what information is contained, and the relationship between different entities in the database, thus increasing productivity for project employees.

## Discussion 4A

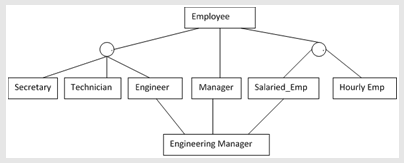
June 17, 2020 5:22:18 PM EDT

EER stands for enhanced entity relationship modeling. This allows for more complex portrayal of entity relationships, however the standard for EERD’s are less defined than for simple entity relationship, thus results may differ based on who creates the EERD. However, using an EER allows for superclass and subclass modeling, as well as some constraints of the superclass and subclass modeling. The two main constraints for specialization and generalizations are disjointness constraint and completeness constraint. Disjoint is a constraint that specifies that an entity can be a member of at most one of the subclasses of the specialization. For example, an employee can be only one of secretary, technician or engineer. On the other hand, completeness is a constraint that specifies that every entity in the superclass must be a member of at least one subclass in the specialization. For example, an employee has to be either an hourly\_employee or a salary\_employee. Thus, while disjoint specifies one or less, completeness specifies one or more.

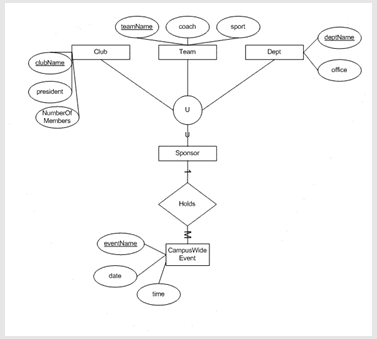
For a given specialization there may be a specialization hierarchy or specialization lattice. For the specialization hierarchy, the hierarchy has the constraint that each subclass has up to one parent and up to one child. This results in a tree structure for the hierarchy. One example of a specialization hierarchy for a person is shown below (note that each person can be only one of each of the subclasses):



The specialization lattice has the constraint that a subclass can have more than one class/subclass relationship. This results in the subclass inheriting all predecessor classes up to the root class. An example of a specialization lattice for an employee is shown below (note that an employee can be an engineer, manager, and salaried\_emp if they are an engineering manager):



Another thing that can be displayed in the EER is something called a category. A category is a union subclass which contain otherwise distinct entity types. For example, the owner of a property can be a person, company, or government, or a sponsor for a university event can be a club, team or department. An example of an EER showing the sponsor category is shown below:



Unfortunately, since the EERD does not have a standard format, there are no database design tools which will do that work for you. To create an EERD, it is best to use a more generic diagram creation tool, perhaps draw.io or powerpoint.

## Discussion 4B

June 17, 2020 5:07:00 PM EDT

… I do not have any personal experience with EER modeling through work or my personal time, so I will elaborate on the pros and cons for EER modeling. To start, some of the pros of EER modeling include creating more detailed relationship models, with specializations and generalizations, disjoint and completeness constraints, specialization hierarchy/lattice structures, and union classes like categories. This has the potential to communicate much more information than a standard ERD. On the flip side, some of the cons of EER modeling include the lack of a EER standard notation, as well as the added complexity potentially taking away from the message that would be more simply laid out with an ERD. Thus an EER could be used in lower-level technical meetings where you want to get into small details of a database design, while an ERD might be more suited to high-level, executive meetings where getting the general idea across is preferred.

From an EER, it is possible to create an ERD from the requirements. The level of detail will be of lower depth, and not all the constraints will be modeled, simply because the level of detail in an EER is higher than in an ERD. However, if you can get the point you are trying to make across with a similar ERD, sometimes you can get away without modeling all the information in the EER.

Alternatively to the ERD, you can also make a UML diagram which will convey some of the same information. One similarity between the ERD and the UML diagram is in the notation for an entity. Each entity is divided into its own area and each of the attributes for the entity are laid out. UML diagrams also specify some relationships between entities, such as the cardinality and subclasses/superclasses. However, the UML diagram will not specify things like foreign keys or foreign keys, and will also include the associated methods for the class/entity. Thus I would imagine that ERD’s are more useful for database design, while UML diagrams are more suited to planning object oriented programming type projects.

UML may be a good fit for some of the conceptual database design phases, but poor fits for other database design phases. For example, UML may suffice for the physical database design. However, when it comes to logical design, it may be challenging to display the conversion between data structure to physical memory. However, for the physical database design, UML may suffice. For the different database design phases, UML may suffice, but it less than ideal.

## Discussion 5A

June 24, 2020 6:06:38 PM EDT

 Relational databases are based off the relational model of data, which is based off the concept of a relation. A database can be represented as a collection of relations in the relational model, where each relation is a table of values, with each value in a row being assigned to a key in a tuple. Each tuple in a relation is unordered, greatly simplifying the process for adding and deleting tuples (as if the tuples were ordered you would have to keep track of the index of each tuple in the relation). Further, duplicate tuples are not allowed in a relation since it violates the specifications of the relational integrity, specifically the constraint that no two tuples can have the same primary key values at any relational state of the database.

There are a few instances where a tuple may have a NULL value in a relation. NULL may indicate that no value has been specified, one example may be a new customer on a site may have not set up their profile completely. Another instance where NULL may appear in a relation is if the value is unknown, perhaps in the case of an optional field that was not filled out, or in a result column where the result is still being computed. Another instance where you may see NULL in a relation is if the value is not applicable, perhaps in the case where you have a relation that stores car sales, one value in the tuple may be cosigner, however not every car needs a cosigner. These are a few cases where NULL may be a valid attribute in a tuple.

There are some constraints to relations that must be considered. The first is the entity integrity constraint, which specifies that no primary key can be null. If the primary key was null, then the tuple could not be identified, hence why the entity integrity constraint exists. Another is the referential integrity constraint, which specifies that the primary key of a parent relation migrates to a child relation as a foreign key to maintain referential integrity between the two relations. If there was no foreign key on the child entity, there would be no way to track the parent in the parent relation, which is why the referential integrity constraint is important to tracking child entities in a relation.

The foreign key is a referencing relation that references the primary key of the referenced relation. For example, if you are selling cars, you may have a relation that describes each car sold, and a relation that describes each of the customers. If a car is sold to child entity customer, then the customer would get the primary key of the car sold added as a foreign key. This allows the database user to track which cars get sold to which customers.

## Discussion 5B

June 24, 2020 6:07:19 PM EDT

As someone new to databases, I have no experience with relational database constraints, no experience with relational algebra, no experiences with SQL, and no experience with query optimizers at work. Thus there should be much to learn for me as I progress through this class.

In lieu of applicable database experience, I will discuss some places in the database project where I will likely use some of the various relational database constraints. For reference, I am planning on doing a database project which will store team and player statistics for a hockey league, and in preparation for the database project I will try to have examples which apply directly to my personal project. There are four types of constraints:

* Domain Constraint: The domain constraint specifies a data type and range associated with a domain. For example, in hockey a player must wear a number between 1 and 99. Thus, in the field for player number, the value must be an integer between 1 and 99. For first and last name for each player, we can specify a constraint of 30 characters, since names do not often surpass 30 characters.
* Key Constraints and Constraints on Null: Key constraints include the constraint that no two tuples can have the same unique identifier or primary key. To satisfy this constraint, for entities such as player and team, on player/team creation each entity should be assigned a unique key as an identifier to satisfy the key constraint. Some attributes may not accept a null value. Attributes relating to statistics for a given player for a given season should not accept a null value. Statistics like games played, goals, penalty minutes, etc. should all have numeric values of 0 or greater, and keys such as these should be initialized to 0 on player creation.
* Entity Integrity Constraints: The entity integrity constraint specifies that no primary key can be null. As discussed above, as each player/team is created, a unique identifier should be created for the player/team, and specifying that on creation will maintain the entity integrity constraint.
* Referential Integrity Constraints: The referential integrity constraint specifies that the primary key of a parent relation should migrate to the child relation as a foreign key to maintain the referential integrity between the two relations. One example of this specific to my database project is each player (child) tuple should have a foreign key specifying which team (parent) they are a part of. This will make it simple to aggregate which players play for which team.

## Discussion 6A

July 4, 2020 6:21:03 PM EDT

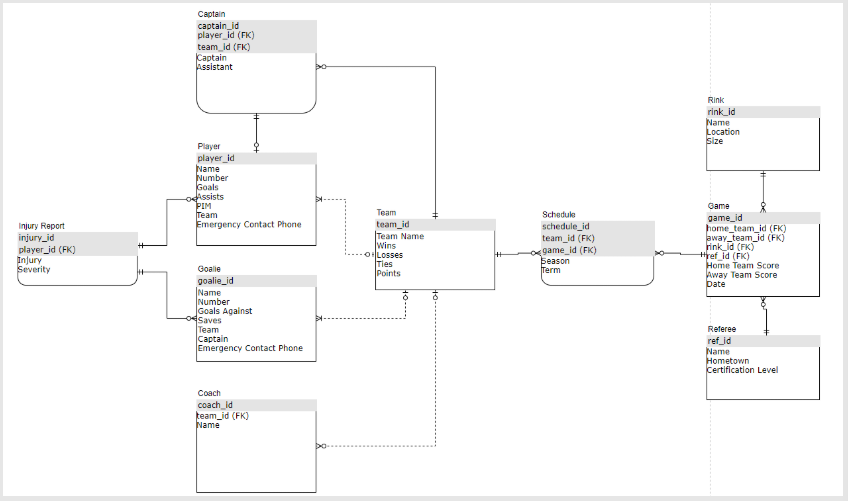
The title of my semester project is ‘Hockey League Database’. The inspiration for this is my own hockey league website, which is likely just a front end for a database similar to this. Currently I have 10 entities/tables to handle things such as players, teams, games, etc.

There were some decisions that I made for my database that could have gone one way or another, for example I specified that a player was different from a goalie (goalies are weird anyway, and their statistics are different than player statistics, so I just made them their own entity). I also specified that players, coaches and goalies could exist independent of a team (for example, free agents or fired coaches, etc.). I also specify an injury report for a player or goalie with player\_id as a foreign key. This player\_id can be for a player or goalie, so the primary keys for both should be independent of each other. Each team also has captains, who can be either the captain or an alternate captain. Goalies are not typically captains, so I ignore that edge case in this scenario.

Other decisions that I made were to use schedule as a way to manage the many to many relationship between team and game. This was an elegant solution to ccreating the schedule with as few entities as possible. For simplicity in this implementation, for each game I assigned only one referee (when in fact hockey games typically have 2-4 referees) just so I did not have to manage another many to many relationship. Further iterations of the database design may include a many to many relationship between referees and games, however for now we can just consider the referee entity to be the “head referee” and allow that person to manage the other linesmen. Each game is also assigned a rink.

The design tool that I have been using thus far is draw.io. Draw.io is a generalized diagram creation tool which has lots of the things you need to create ERD’s and a nice intuitive interface to create diagrams. The flexible interface for draw.io also allows you to create diagrams in Chen’s notation, which was useful for the midterm. For the final iteration of the database design project, I likely will have to create something in mysql workbench or another sql tool, however I can do that research and learning later in the term.

See my design for my Hockey League Database below:



NOTE: This ERD was appended in a follow-up comment

## Discussion 7A

July 8, 2020 8:40:32 PM EDT

A successful database design will be one such that there is no insertion, deletion or modification anomalies present in the relations. An insertion anomaly can occur in the case where it is impossible to insert a row into the database, for example if two relations are required to have a reference to one another, it would be impossible to insert just one or the other. For example, if a team in a sport league database is required to have players, and players are required to be assigned to a team, then neither will be able to be inserted to the database. Deletion anomalies occur when deleting a single entity has unintended consequences to the rest of the database. Using a similar example from above, if a team is deleted and it went and deleted all the players as well, that would be a delete anomaly (since the players could find their way to other teams after the deleted team is disbanded). Update anomalies occur when changing one attribute could change attributes for other entities. For example, if a player changed teams, an update anomaly could be if the database updated the team value for each player on the team. These anomalies are considered bad because they corrupt the data, and we never want to corrupt the data in our database.

Null is a database value that should be avoided if possible. Some disadvantages of storing null include requiring extra data storage to track nulls, null may not work with aggregate operations like sum, avg, count, etc. and that null may have different interpretations. Spurious tuples can occur in join operations where database design does not enforce that relational schemas can be joined with equality conditions on attributes without creating the spurious tuples. Spurious tuples can cause issues in data aggregation and downstream applications.

One challenge I faced when performing normalization is in determining which entities are dependent on which primary keys. For example, in the homework it can be tricky to determine if the OwnerNo belongs to the PropertyNo key or both the PropertyNo and the ClientNo keys. This is important in performing 2NF, and can cause challenges down the line for the normalization. Normalization is important because it is a process to review and validate relations as well as to eliminate duplicate data. Each normalization must be performed in sequence, since the operations are not transitive. A mistake in 1NF can cause the solution from 3NF to be invalid, for example. The only normalization form which does not consider functional dependency between key and non-key attributes is 1NF, where you split data with multiple values into its own relations.

## Discussion 7B

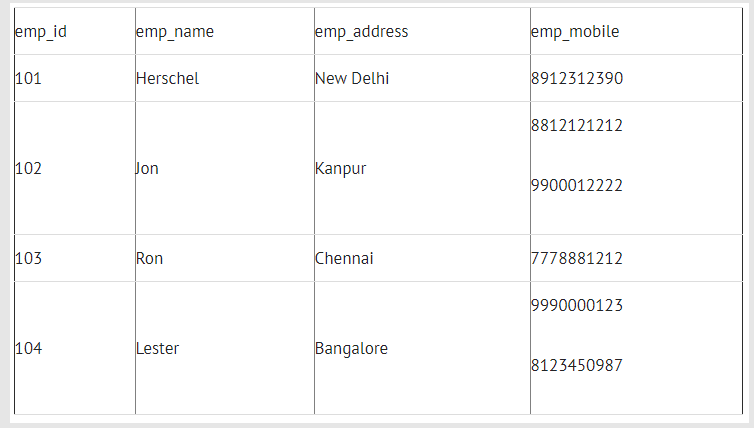
July 8, 2020 8:45:00 PM EDT

Top-down methodology is generally more effective than botton-up methodology, since it captures the entity types and their associated attributes, as well as their relationships based on the requirements. Doing a bottom-up approach for database design can result in repeated attributes as well as making relationships more difficult to define.

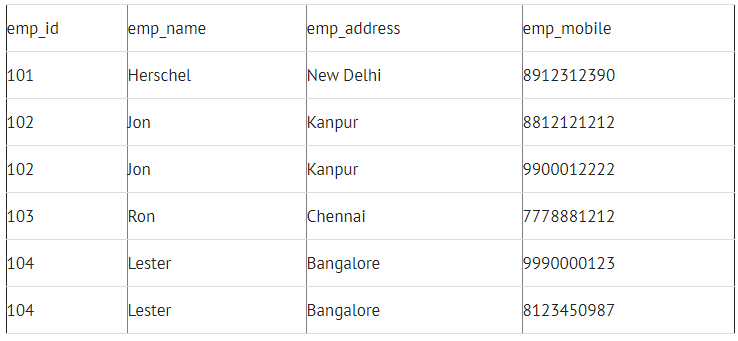
As I am a novice in database design, I do not have any experience in functional dependencies or relationship schemas from earlier schooling or work.

I was able to find some examples of 1NF, 2NF, 3NF online that I can share with the class. For reference, the entire article that I read was here: <https://beginnersbook.com/2015/05/normalization-in-dbms/>

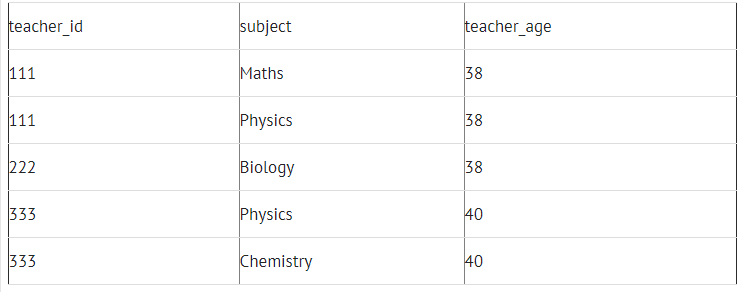
For 1NF, suppose a company wants to store employee contact information. One example of this is in the below table:



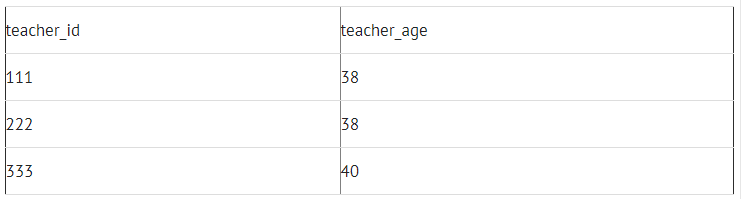
However, we note that some of the tuples have multiple values for emp\_mobile. This can be handled in the 1NF stage, and a resulting database design would look like:

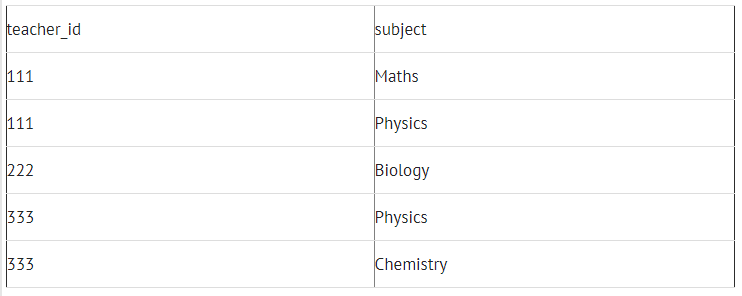


For 2NF, suppose a school wants to store information about their teachers. One example of a table to do this would be like this:



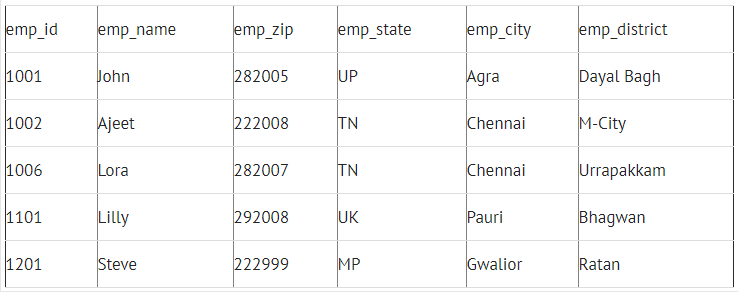
However, the teacher age does not depend on the subject taught. Thus, we can break this down into two different relations:





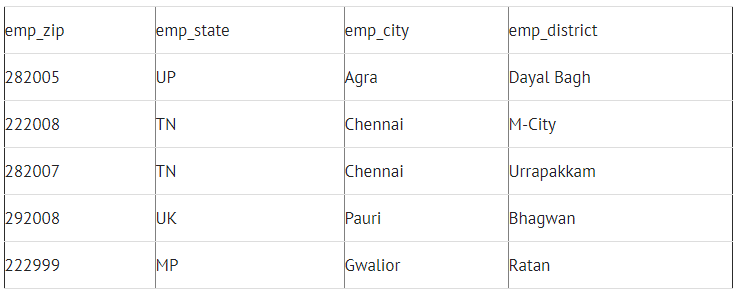
This eliminates some redundancy in the non-2NF table.

Finally, for 3NF, we note that some of the values in the table may not depend on the primary key for that entry. One example of this is in the following table which stores employee address information:

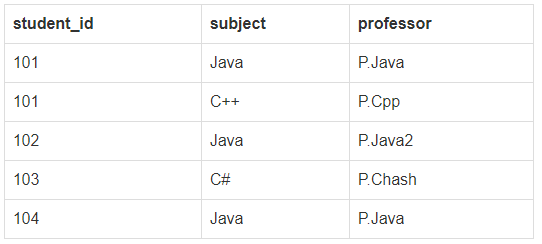


However, we note that the emp\_state, emp\_city, and emp\_district all depend on the emp\_zip. Thus, we can split this into two tables to reduce data redundancy. See the two tables below for how to do this:





BCNF is even stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y,X should be the super key of the table. One example of this would be the following table:



This table is not in BCNF since each professor is dependent on each subject. Thus we can break out the previous table in to the following two tables:





Thus these are examples of 1NF, 2NF, 3NF and BCNF.

## Discussion 8A

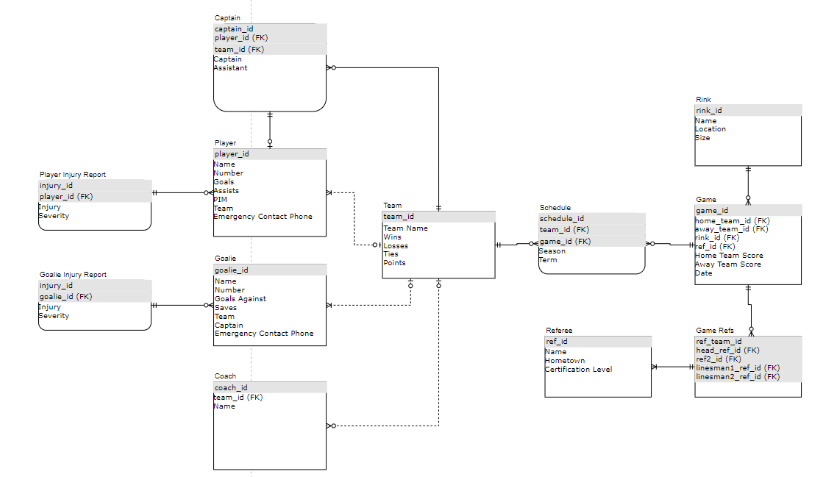
July 15, 2020 7:23:28 PM EDT

MVD, or multivalued dependency is a dependence between attributes in a relation such that for each value of A, there is a set of values for both B and C, where the set of values for B and C are independent of each other. This can often arise when a table is taken out of 1NF, which disallows attributes with multiple values. This happens often when two independent 1:M relationships are mixed in the same relation.

While MVD’s can occur in relations with two or more columns, this is not always the case. One example of a relation with 2+ columns and no MVD’s is shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | First | Last | Age |
| 0001 | Brian | Loughran | 25 |
| 0002 | Nico | Hischier | 21 |
| 0003 | Jack | Hughes | 19 |

Some minor changes have been made to my original ERD for my database project to ensure normalization in 3NF. I believe that my database happens to be compliant with 5NF, however that was less of a conscious effort than it was a design choice. It seems that most databases are normalized to 3NF as is industry standard, so if my database was not compliant with 4NF or 5NF, I would not consider it to be too much of a worry.



Sometimes it is useful or necessary to add or change index structures to create a cluster for improved data access times. Indexes can provide a quick way to access specific rows and avoid a full scan of a table. This process is known as denormalization, however one should be wary of performing this process. Denormalization can cause data inconsistency and result in poor table design. It also may result in more redundant data and increase cost of tracking and handling duplicates and spurious data.

I do not have any personal experience with denormalization.

Some reasons a relation may not be in 1NF include the need to store multivalued attributes or resolve 1:M relationships in a single table. A table may not be in 2NF if an attribute does not depend on the full PK, a common issue with those new to database systems. A table may not be in 3NF if a primary key is not identified for a relation, such as the way that zip code can represent town, state, etc. A table may not be in 4NF if attributes within the table have an M:N relationship, and a table may not be in 5NF if a designer does not recognize a join dependency.

## Discussion 8B

July 15, 2020 7:25:07 PM EDT

There are many options for mapping single-relation and multiple-relation mappings from the enhanced ER model.

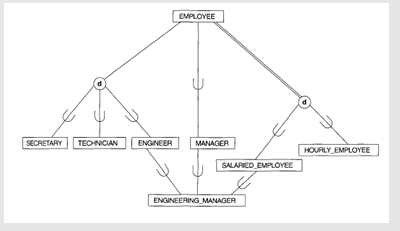
One option is to create the superclass relation and all subclass relations first with their attributes and then migrate the PK from the superclass relation into each subclass relation. This is a simple solution, but the PK from the superclass can be passed down multiple levels, which can cause confusion and take the database design out of 2NF.

Another option is to create a subclass relation and add all superclass attributes to the subclass relation. This will result in fewer relations, but will open the database up to the potential for duplicate data and will also take the database out of 2NF.

Another option is to create a relation which is a union of all the attributes from all subclass relations with a type attribute. This is for specializations with disjoint subclasses. This will result in far fewer relations, however will create many null values and you have to track different flags for each entity.

A final option is to create a single relation which combines all attributes from the superclass and subclass relations with a set of Boolean flags to indicate whether the tuple has the specified types. This is an extension of the previous option for overlapping subclasses, and has the added flexibility of maintaining overlapping attributes, but will again create many null values and force you to track flags for each entity.

These are all valid options to handle the mapping of the enhanced ER specialization lattice for engineering manager from figure 4.6 of the textbook (shown below):



There are two options of the bunch that stand out to me as a way to map the subclass/superclass relationships for the given figure. The first option discussed is typically a good option for relationships without too many levels, and mapping the primary keys from employee all the way down to engineering manager is only two levels of mapping. The fourth option discussed is also a good option due to the overlapping subclasses, since engineer, manager, and salaried employee all overlap into engineering manager. However, due to the increased complexity of the fourth option and all the null values introduced, the first discussed option is likely best for the given superclass/subclass relation. Option one is a single-relation mapping, thus each relation in the eventual class diagram will match each entity in the specialization lattice in figure 4.6.

Some of the challenges associated will be mapping each of the foreign keys from each of the relations when using the first discussed option. Engineering manager will have two foreign keys that are also primary keys, while engineer, manager, and salaried employee will each have one. Another potential challenge is in collecting all associated keys for the engineering manager. This will require a join between each of engineering manager, engineer, manager, salaried employee and employee. While collecting the attributes for engineering manager in this way may not be the most efficient, it feels much cleaner than option 4 where the data may have many more null values, flags to point which keys are valid, etc.

Using UML class diagrams for the conceptual database design phase may result in some challenges for mapping class to relational mappings. One way to get around this is to ensure that the primary key mappings are shown as attributes for the subclass. This is one way to show the mappings. Another way to get around this potential pitfall is to simply use an ER diagram in IE notation with labels on the relations to show the relationships. Again, some of the information is lost from the EERD, but this can provide a solution which will save as much data from the EERD as possible.

## Discussion 9A

July 23, 2020 6:16:59 PM EDT

Entity integrity and referential integrity are used in SQL and are important in maintaining the data structure specified by the database designer. SQL allows implementation of the entity integrity by using the primary key clause. Meanwhile, referential integrity is upheld by using foreign key.

I have not done any work with SQL, DDL\_Does, etc. outside of this class, so I hope to learn these processes over the course of doing the homework. I have done some work with NoSQL, however, although the queries are very dissimilar from SQL it seems.

The RDBMS that I am planning to use for HW4 is PostgreSQL. PostgreSQL is an open source relational DBMS with good documentation that I believe will be a good fit for the homework and database project. I came to this conclusion by comparing relative ease to learn with some other RDBMS platforms like MySQL and Oracle, determining that PostgreSQL would likely be the best for me and my scenario. PostgreSQL is new to me, but it appears to support all standard SQL syntaxes.

I have done some date and time manipulation at work. I have found that it is very annoying when the source data has different formats for date and time. I find that it is best to store date and time the same throughout the database and do the conversion to the format that you want on the client side rather than have different formats stored in different places on the database side.

For performing SQL queries to get information from multiple tables based on relational database schemas I prefer to use IE notation rather than Chen’s notation. As I discussed previously in discussions, I find that IE notation is easier to parse, neater, and more compact than Chen’s notation, making it just slightly easier to determine how the query should work. Now that I am learning how primary and foreign key mappings work in RDBMS, I am starting to see how the key mappings from IE notation can help with creating queries.

## Discussion 9B

July 23, 2020 6:17:27 PM EDT

SQL statements are built of formatted blocks of commands which pull data from the database. The SELECT clause lists the attributes/functions to be retrieved. The FROM clause specifies relations needed in the query. The WHERE clause specifies the conditions for the selection and joins with records specified in the FROM clause. The GROUP BY clause specifies groupings. The HAVING clause specifies a condition for the grouping selection, and the ORDER BY clause specifies an order for the query result. The only clauses that are required are SELECT and FROM, with the rest being optional.

SET operations such as UNION, INTERSECT and EXCEPT are similar to union operations discussed previoiusly in the course. UNION combines the result of two queries, INTERSECT is what two queries have in common, and EXCEPT includes everything in the first query that is not in the second query. An example would be if we had two zoos, where zoo1 had monkies, lions and penguins, and zoo2 had penguins, seals and dolphins, we can do the SET operation on the two zoos. UNION(zoo1, zoo2) would give monkies, lions, penguins, seals and dolphins, INTERSECT(zoo1, zoo2) would return penguins, and EXCEPT(zoo1, zoo2) would return monkies and lions. This concept is fairly simple and has been elaborated upon in earlier lectures.

The RDBMS that I am using is PostgreSQL, which seems to support each of UNION, INTERSECT and EXCEPT. This saves me the hassle of creating a workaround for all of the SET operations.

Subqueries in SQL are cool because they can introduce more complex logic to the SQL statements. The cons, however, include increased complexity for the queries. Subqueries allow the database manager to break out the logic of their SQL commands into discrete parts and create a logical order in which to process queries. Join is also a way to do this, where the database manager can do a query with embedded logic and join the results to create the same result as a subquery. The logic and thought process is similar between subqueries and join, however the syntax is different. For complex queries, I may lean toward trying subqueries first due to the logical nature of the query language using subqueries, but without any experience trying either, that opinion is open to change at any time.

## Discussion 10A

July 29, 2020 4:17:14 PM EDT

I have very limited experience in SQL outside of HW4, thus I plan to lean on the lectures and topics that have been discussed thus far in this course. I have found the syntax of SQL relatively easy to pick up; when creating tables and inserting tuples the syntax is very easy to follow exactly what you are doing. This is also my experience for simple queries, however as the queries get more complex (especially for the division part of HW4) I have found that sometimes the syntax can be a little confusing, and it may be difficult to parse exactly what is happening. This is the case for many programming languages as well, however, thus I am hopeful that I can learn more with time.

I have no prior experience with SQL, DML, sub queries, joins, views etc from school, which was a big reason that I decided to take this course. At work, I have done just a tiny bit (select \* from table) to query some data that was in a postgresql table, anything more complicated than that I would need help from one of the database experts in the group. In my free time I built out a NoSQL database, however that has very different syntax, and not much carried over to SQL.

I have absolutely no experience with with query optimization in my prior education or work experience. The NoSQL database that I worked on did not require any query optimization either. Some strategies to try during query optimization include compressing the data into fewer tables, repeating data, creating materialized views, etc.

I have no experience with handling dynamic SQL and SQL injection. I was not made aware of these concepts before the start of this course even. I would imagine that my colleagues consider these types of things, especially the database experts since we work often with classified data. I am interested to poll my colleagues at the conclusion of this course and quarantine on some of the methods they use to further my knowledge on the topic.

## Discussion 10B

July 29, 2020 4:56:56 PM EDT

I do not have any current experience in implementing complex business rules with triggers, stored procedures, functions and packages. Some possible examples of business rules (I am using Amazon as an example) is to ensure that a product is in stock before an order is created. Or ensuring that a shipping address is accessible by Amazon shipping before placing an order could be a business rule. Computing the estimated shipping time can be done using a stored procedure based on the current product location and product destination.

For my own database project there will be some business rules that will need to be put in place. My database project is a database system for a hockey league. I have a few business rules in place to ensure that a certain entity exists before it can be created. One example is that a player must exist before a player injury report can be created. Another example is some limitations on the data that can be passed to the database, for example the rink size must be either ‘standard’, ‘olympic’, or ‘half-sheet’, captain status must either be ‘assistant’ or ‘captain’, etc. I have not used stored functions yet, so all of my business rules are built using triggers currently.

SQL injection is a security attack against a database where the attacker enters SQL code into a web form box to gain access to database resources maliciously. Injecting malicious data can have unexpected consequences for the database behavior. the best way to prevent SQL injection attacks is to control the type and number of characters accepted by input, ensure access control over sensitive tables and columns, and to use security software such as AppScan, Netsparker, Fortify Software, etc. to identify application vulnerabilities.

I do not have any experience with object-relational features in my professional, education or personal life. Object relational features allow for expanded data types to be inserted to the database, such as nchar, varchar2, date, bfile, etc. Collection items such as varray and table can be added as well. Other things that can be added are user-defined data types, objects with attributes and member methods, LOB’s such as BLOB, CLOB, etc. While this can allow for greater table flexibility, it can also increase data complexity, and take the relation out of 1NF, 2NF or 3NF, depending on the data type inserted.

Some challenges that I observed working on the database project was ensuring the relationships between each of the relations, managing the migration of foreign keys, applying appropriate constraints to my relations, and ensuring that the data was inserted in the proper sequence. This was a lot of trial and error, but getting things wrong and seeing error messages helped me a lot to learn further how to create databases in a way that makes sense and will work.

## Discussion 11A

August 7, 2020 6:10:30 PM EDT

Unordered files are the simplest type of file organization, records are generally ordered in the order they were uploaded in the system, which allows for very fast insert. The disadvantage with unordered files is that search is usually much slower with unordered than ordered, where search can take O(n) time. Sorted files can provide faster search, especially using BST which can reduce search time to O(lgn), however will take longer to insert records. Using static has files can provide very fast insert (almost as fast as unordered) and fast search in some conditions. Some drawbacks to static hash files include additional complexity to implement and can show some inefficiencies for large databases.

I have no experience with database file structures, however I gather that depending on the size of the database you may want to try different things. For small databases, like the database project, it may be easiest to just use an unordered file. For smaller databases where search time is critical, a hashing technique can be employed to improve search time. For larger databases, it is likely that an ordered file would be best.

Indexes can be extremely useful in data retrieval, however the increased overhead of the indexes needs to be considered before implementation. Indexes should be avoided for small indexes or if queries comprise 10-15+% of the rows. Indexes should also be avoided for tables where the rows are updated often, as the indexes will also have to be updated. However, if these conditions are met, then indexes can greatly improve data retrieval. Indexes are even more useful in data retrieval when the query involves a JOIN or WHERE clause.

I have no experience with special indexes like bitmap or functional based or reverse indexes. I learned that function based indexes are appealing when handling computationally expensive functions, and can be precomputed and stored before queries. Bitmap indexes are helpful in data warehouse applications and are especially useful for queries that contain EXIST, UNIQUE or GROUP BY.

## Discussion 11B

August 7, 2020 6:39:04 PM EDT

B- Trees are trees that can have multilevel indexes which can be useful in searching for records in a data file. B- Trees are always balanced, which reduces wasted space by deletions. Insertion and deletion are generally simple, but can be complex in cases where you have to enforce the balance of the tree. B+ trees are a variation of B- trees, where the B+ trees only store data in the leaf nodes, and store keys only in the internal nodes. This makes B+ trees the generally preferred choice since records are linked like in a linked list, making search faster. B+ trees also have easier deletion than B- trees.

The RDBMS that I used for the database project was postgresql, which supports several types of indexes. Postgres supports B-tree, hash, gist and gin indexes. Postgresql also supports multicolumn indexes partial indexes. Some scenarios that I would use indexes is a B-tree for ordered records, or hashing for fast insert/search. I have not personally used any of these indexes for the database project.

I have no experience with improving database performance, however some strategies that I would use include things like indexing or reorganizing tables. Indexes can be used to increase search speed and performance for databases, especially for queries that use a lot of joins or have computationally expensive steps.

CRUD matrix is a good tool for understanding which functions perform which operations on which database entities. This can organize the functions and entities in the database, and show what operations access what data. CRUD does this in a graphical way that is easy to parse. Query by example is a useful tool which allows a user to retrieve data from a database using a GUI. This helps individuals with less experience with databases to specify what they want to query, and allows them to begin to learn. Most DBMS tools have some form of a query by example tool to help the users.

Throughout the database project I learned a lot about database implementation in SQL, which is exactly what I set out to do in this course, so for that I am thankful. In terms of strengths, I think I have a very good grasp of how to design a database in a way that makes sense with foreign keys and primary keys and relationships. One possible weakness still is with more complicated SQL queries, for example the query from HW4 was difficult for me. However, as I practice more I am getting better, and some of the queries included in this report were relatively complex.

# REPLIES TO CLASSMATES' POSTS

## Discussion 1A

June 1, 2020 8:25:15 AM EDT

Haha, interesting is a good word for it!

For me it has been a lot of reading and independent research, as well doing my best to avoid database work in general. It will be nice at the conclusion of this course to be able to feel more comfortable with databases and be a generally more balanced software engineer.

June 1, 2020 8:29:27 AM EDT

Hey Andrew, thanks for sharing,

I work at Pratt & Whitney, so it is nice to see another aerospace person in the class! I really like working at the intersection of the aerospace and software industries, as I think they are both such interesting fields to work in. It sounds like your role is relatively similar to mine, so you must work on pretty interesting stuff too.

Are you allowed to share what planes you are working on? I know with clearances sometimes you can't share, but I would be interested to know if you were allowed to share.

## Discussion 1B

June 1, 2020 8:31:54 AM EDT

Hey Robert, thanks for sharing. I was curious about the Android development course, I was considering taking it myself. Do you remember what software you to develop your app? And do you remember what database system you used? I have been doing some benchmarking for possible software packages for an android app, so I would be curious to hear what JHU is teaching.

## Discussion 1C

June 1, 2020 8:42:22 AM EDT

Hey Adrienne,

Thanks for sharing, I think you did a very good job laying out the solutions to the questions being asked in a clear, concise way. I also appreciated you explanation of the different schema languages and how one package (like SQL) can encompass multiple schema languages. When reading about all the languages I was a little confused because I knew things like SQL existed and could accomplish queries specified by more than one of the schema languages.

Anyway, thought that was interesting, thanks for sharing.

## Discussion 2A

June 4, 2020 6:19:21 PM EDT

Hey Maria, thanks for sharing this detailed post, I think you lay out the challenges of the ERD very well. It is also heartening to see that I am not the only one that considers the relationships to be the more complicated part of the diagram. Managing the relationships between each of the entities in the ERD requires careful planning and thought.

You also make a good point that you can increase the complexity of the diagram quickly by drilling down into the different types of entities. Your example of splitting the EMPLOYEE entity into QA and developers would likely give different relationships between the two entities. Another example might be splitting DEPARTMENT into the different real life departments. The R&D department, then, would likely have different relationships to entities than the HR department. One suggestion I might make in this case is to manage the scope of your ERD by keeping each of the entities vague. This will allow for high-level diagrams of complex systems, but will also require careful thought of the relationships between the large entities, since the relationships will have to encompass each of the subsets of the entity.

## Discussion 2B

June 4, 2020 6:35:33 PM EDT

In response to your second paragraph:

My work has moved from waterfall to agile for almost all software development projects. While waterfall often has an advantage when the requirements are very well defined, we often discover not even the customer knows exactly what they want at the offset of the project, and that their needs evolve over time. Agile gives us a framework to both prioritize the most important work in a quick way and also to be flexible enough to meet those always changing customer demands. Coupling agile with schedule tracking tools such as burndown charts also allows us to also predict schedule risks long before the actual deadline, giving us time to make course corrections (like adding more resources, etc.) to meet deadlines. Agile is also a flexible framework, so you can add as much or as little oversight as you prefer for your team. So I would probably recommend agile for most projects, however there are some edge cases where waterfall might work as well.

## Discussion 2C

June 4, 2020 6:51:01 PM EDT

For reference, and based on some brief research that I have done, I think I am going to try postgresql for my project just looking at some of the pros and cons of each postgresql vs. mysql ( <https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres#:~:text=Postgres%20Advantages%20over%20MySQL,more%20closely%20to%20SQL%20standards.>). For installing/learning postgresql, I might reccomend this tutorial:

<https://www.postgresqltutorial.com/>

It seems pretty well broken up and laid out, so I will follow it until I have an issue. Best of luck to everyone creating their own databases this semester!

Sorry for the multiple posts. I have been doing some research on different tools to use for the project, and I wanted to share any helpful information that I found. I was having a tough time finding free tools to create the ERD for the project, many tools like lucidchart, sqldbm, quickdbd, etc. offer only free trials, and if I can get away with not paying for a software I will often try to do just that. I have used draw.io in the past to create general diagrams, however I did not realize for quite a while that draw.io contains all the tools needed to create the ERD in IE notation. Draw.io even comes with template ERD's to help you get started! This youtube video is a good example of how to get started in draw.io creating the ERD using a template:  <https://www.youtube.com/watch?v=DpZTx5wfXyk>

June 4, 2020 6:49:56 PM EDT

Hey Ivan, thanks for sharing that reddit link. Looks like they suggest either postgresql or mysql as a dbms. I think I am going to try postgresql for my project just looking at some of the pros and cons of each ( [https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres#:~:text=Postgres%20Advantages%20over%20MySQL,more%20closely%20to%20SQL%20standards.](https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres" \l ":~:text=Postgres%20Advantages%20over%20MySQL,more%20closely%20to%20SQL%20standards.)). For installing/learning postgresql, I might reccomend this tutorial:

<https://www.postgresqltutorial.com/>

It seems pretty well broken up and laid out, so I will follow it until I have an issue. Best of luck with your own project!

## Discussion 3A

June 12, 2020 5:12:02 PM EDT

Hey Nick, thanks for sharing, I think you summarize the differences between Chen and IE notation very well, and I agree that IE notation is cleaner and easier to absorb. I also agree that migrating foreign keys from Chen's notation to IE is tricky, I have been trying to work through the issue for my homework assignment, and some of what you have said about foreign keys was helpful.

As for managing attributes of relationships that don't get converted into intersection identities, I believe the rule of thumb is that the attribute is migrated to the child entity (thus Start\_date would be a part of the employee entity). The professor sent an email that I believe adresses this on 6/9. Hopefully that helps a little!

## Discussion 3B

June 12, 2020 7:07:34 PM EDT

Hey Anh,

Interesting proposition to seperate out more tables for Terms, Locations, Department and Ranks to look up foreign key values. I am not 100% sure that I understand the advantage, however. Is there a use case for changing the keys in the seperate tables? Maybe I just do not know of the reason one would want to do that. Creating those different tables may also make it difficult to find things in the database. For example, if you are trying to find a student record for a particular class, would you then have to look in each Term table? Or if a student wanted to take a class but didn't know what department it was in, would the query have to check each of the Department tables? While I could see how it might be helpful in some cases to separate out into more tables, I can also see situations where it may add complications to the system. Perhaps it all comes down to use case.

Anyway, I think your questions were great, and we had very similar assumptions. Thanks for sharing

June 12, 2020 3:57:20 PM EDT

Great to hear we agree on our assumptions! Always nice to hear some affirmation from another student, gives me a good idea that we are doing it right.

As for the questions, I am sure that as you increase the complexity of the database you can add further questions to ask. These were first pass, off the top of the head for me, and I am sure that if anyone in the class brainstormed long enough they could also come up with some questions for the customer that I had not thought of. Often it can be helpful to have a group of people asking questions to cover all bases.

## Discussion 3C

June 12, 2020 7:12:14 PM EDT

Hey Ivan, thanks for sharing that link, must have missed it in my research! I was hoping that there would not be any assignments where we would have to use Chen's notation, however if there is, I certainly will start by digging deeper into that link. Thanks for sharing that with me

June 12, 2020 7:16:35 PM EDT

I am seeing a lot of classmates leaning toward MySQL workbench to create the ERD diagrams, and it seems to have a bit of a higher learning curve than other tools. I realize a lot of this has to do with it doing a lot of stuff in the background for you, which is great, but perhaps more power than needed for the first homework assignment. For reference, I have been using draw.io which seems to have a lower learning curve (however you lose all the goodness that MySQL workbench does for you in the background since it is just a diagram creation tool).

Draw.io also creates a template ERD for you if you if you don't know where all the different connections and shapes are. You can follow this link to get the template ERD: https://www.youtube.com/watch?v=DpZTx5wfXyk

Just wanted to reiterate that draw.io is also a good tool for creating ERD's. Feel free to use whichever you are most comfortable with.

## Discussion 4A

June 19, 2020 1:19:15 PM EDT

Hey Maria,

Thanks for adding to my post the bit about total and partial properties. That is a good detail that I did not quite get around to mentioning.

And as for lucidchart, that software did not seem to be free. I am trying to confine my search to free tools where possible, so I had not looked closely at the full capabilities. Very interesting that they have the capability though, if we wind up having to make an EERD for this class I will definitely take a look, making that diagram in draw.io or powerpoint would be quite the hassle.

June 19, 2020 1:23:07 PM EDT

Hey Juan,

Just to add to your point on the database design tool to support an EERD, I did a little research on the topic as well and it seems quite challenging to find a database design tool to support the EERD. I could not find any free tools outside of draw.io or powerpoint, which would be quite time consuming to create an EERD in. However one paid tool, lucidchart, seems to have ready-made templates to support something like an EERD. Perhaps something to keep in mind just in case we have to create an EERD (which I genuinely hope we do not have to do) for this class.

Thanks for posting!

## Discussion 4B

June 19, 2020 1:26:47 PM EDT

Great point about the market dominance of ERD and customer meetings, an EERD in that setting may seem out of context for a customer if they are not familiar with them. For some reason in my post I only considered internal meetings, but for an external meeting I would agree that we should lean more towards an ERD. Thanks for that.

June 19, 2020 1:29:26 PM EDT

Hey Ivan,

Thanks for sharing, I think you do a good job pointing out the pros and cons of the EERD. One other con that I could mention is that since ERD's seem to dominate the market today, it may be slightly offputting to a customer to show an EERD. It seems that most customers would expect an ERD, so for external meetings that could be another con for the EERD.

## Discussion 5A

June 26, 2020 12:26:16 PM EDT

Hey Ivan, thanks for sharing, and thanks for keeping your answers brief and to the point. I would up using a lot more words than you and ended up conveying much of the same information. The language you use is much more to the point than what I used.

I also included that on top of the key constraint that it would make no logical sense for two tuples to have the same information. Duplicate tuples would have the same primary key, thus there would need to be logic for which one you want to access. It is simpler, then, to just force the constraint that there are no duplicate tuples (and no two tuples with the same primary key).

## Discussion 5B

June 26, 2020 12:30:05 PM EDT

Hey Nick, thanks for that post. I ran into a similar issue with one of my side projects trying to eliminate old data from a database. I seem to have come up with a similar solution to you, basically scheduling a job daily to crawl the database and remove any old data. I thought it was a little hacky at the time, however it is a little heartening to see that other people are doing the same thing. I would generally prefer if you could set some rules that if the data is however old it will delete itself from the database automatically (somewhat similar to some AWS S3 rules you can set, for example delete after 7 days if not used, etc.), however I was not able to find an option like that for my database. Anyway, I am glad that my examples were helpful to you, thinking about them and actually writing them out were helpful for me too!

June 26, 2020 12:33:43 PM EDT

Hey Ivan, thanks for sharing, this is a pretty good, quick summary of some database constraints.

One thing I might add is that the key constraints also specify that no two primary keys can have the same value. Of course, this makes sense, since if two primary keys had the same value, it would be impossible to query which one you wanted with just the primary key, which is not a good database design. And any value can be required in the table, meaning that it can not be null.

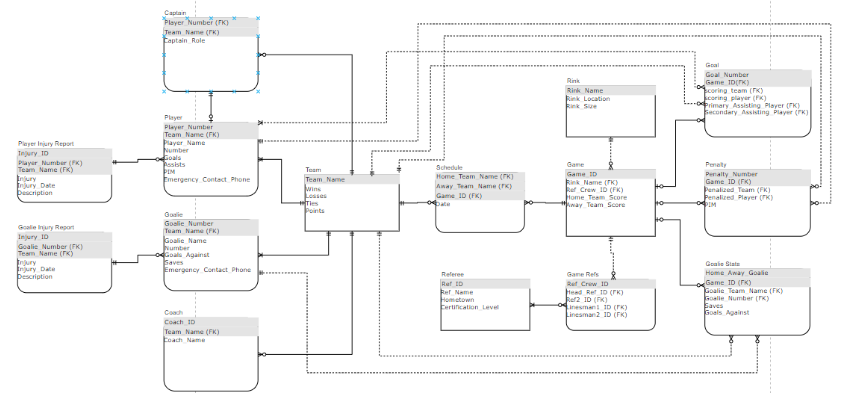
## Discussion 6A

July 4, 2020 11:45:14 AM EDT

Hey Stephen, this is very cool. I am a little shaky on my machine learning knowledge, I am interested in the subject but I have not had a chance to take a class on the topic yet, so I have no real knowledge of machine learning database applications. My first instinct for a dataset like that would be to use as few tables as possible to reduce complexity of the application and hopefully have more efficient data load times. I was wondering if you had any insight into reducing the number of tables to increase efficiency, or if you could make the queries as efficient or more efficient with the increased number of tables. Any thoughts one way or another?

August 6, 2020 5:38:14 PM EDT

I dont know if anyone is checking this, but I made some major changes to my ERD. I also realized at some point in the semester that I needed 12+ tables, for some reason at the midterm exam I thought it was 8+. But anyway, for completeness, attached is my final ERD:



## Discussion 7A

July 10, 2020 10:41:03 AM EDT

Hey Anh, this is a good post, covers the discussion topic well. I had just minor additions to your paragraph about normalization, you seem to have covered the most important points. One other thing that I included in my post was that 1NF, 2NF and 3NF MUST be performed in order, since the transformations are not transitive. Performing 1NF then 3NF then 2NF may result in an incorrect solution in some cases, even if each of the normal form transformations are done in the correct way. Other than that, you cover everything well, thanks for sharing.

## Discussion 7B

July 10, 2020 4:16:29 PM EDT

Hey Ivan, you make a good point about figuring out which attributes have functional dependencies on which keys. Sometimes it can be difficult to determine, and I don't have a perfect solution for you. One thing that helped me in the homework was to not focus on the keys and which attributes belong to them, but rather to focus on each attribute and decide which keys are absolutely necessary to identify them. For example, in most cases city town and state can be determined from just zip code. Looking at the attributes helped me in the homework rather than looking at the keys and determining the attributes from there.

## Discussion 8A

July 17, 2020 12:33:23 PM EDT

Hey Juan, you make a good point about normalizing past 3NF. I also agreed that I did not think that I needed to go past 3NF since it would cause my ERD to need more joins and also lose some of the separation logic. The main benefit of normalization is that it separates out data that does not need to be together and removes a lot of redundancy and null values from the relations, however the downside of this is more relations in the database. So there are pros and cons of normalization, and I feel that the "sweet spot" for normalization is right at 3NF.

July 17, 2020 12:22:08 PM EDT

Hey Maria,

I agree with many of your points. Normalization is typically good since it creates readable, manageable data, however normalization has the downside of creating more tables, which could reduce database retrieval speeds due to the extra joins that come with more tables. Usually a database can handle that, however in some cases this can be a restriction.

I also agree with you that I do not see the need to do BCNF, 4NF and 5NF; sometimes it feels like these normalizations take some logical structure away from the data. They are also relatively rare cases that do not come up at all in some databases, so oftentimes a table that is normalized in 3NF is also normalized in 5NF.

Thanks for sharing

## Discussion 8B

July 17, 2020 12:36:53 PM EDT

Hey Juan,

I chose option A for mapping the specialization lattice from figure 4.6, however I can see how you could come to the conclusion to use option C. I used option A since it preserved the logical separation of each of the entities, with the drawback that entities further down in the lattice had to have composite keys. I chose this because it made more logical sense to me to have composite keys than flags in a relation to determine whether the entity would have a specific attribute. This would also cause the relation to have a lot of null values. However, any of the four options discussed can work, and I like how you justified using option C.

July 17, 2020 12:42:20 PM EDT

Hey Nick, good post detailing the different options for putting EERD's into a class diagram. You lay out the pros and cons for each nicely, and do a good job justifying why you chose one of the latter options. I also liked that you actually did the mapping for each of the options laid out, the relations

as set up look correct to me (sans the attributes for each entity, however those were not specified in the problem).

Thanks for sharing

## Discussion 9A

July 26, 2020 8:24:17 PM EDT

Hey Anh, you provide some good perspective on what you do personally day-to-day at your company with databases. I feel that depending on the maturity of the project, a database engineer may do very different things in SQL. Early in the project, the engineer will likely be using CREATE to create the relations and add constraints, while later in the project, which I imagine is where the project you are working on is, there will be much more like you described with SELECT, INSERT and ALTER.

Also interesting that you are using MySQL. I also looked into MySQL for my database project, but some forums on the internet seemed to prefer postgreSQL. I am sure that both are entirely sufficient for the database project, and I wish you the best of luck on it!

July 26, 2020 8:20:01 PM EDT

Hey Nick,

I must admit, I was quite surprised to hear the the customer you work with typically provides the DLL's for you to make changes to their data models. That is a level of detail that is usually abstracted for the customer. If they have that level of expertise to create the DLL's, it feels that they might be better served doing the work themselves rather than going through the trouble of having a contractor execute the statements. However, there can be some benefit gained from having someone else look over things like DLL changes and ensuring naming conventions and correctness.

## Discussion 9B

July 26, 2020 8:40:11 PM EDT

Hey Maria, I thought you did a great job giving examples that relate to your particular database project. Many times operations such as these are hard to learn without good examples, and I think that by assigning the example to something that you know well and has some consequence to you that the example becomes much better and the operation then becomes easier to learn. I thought you INTERSECT example was excellent, and made immediately clear how it works.

Thanks for sharing that example

## Discussion 10A

July 29, 2020 4:26:12 PM EDT

Hey Adrienne, interesting to hear about your experience with SQL, it seems that you have much more experience than my very limited understanding of SQL and databases in general before this course. You bring up indexing with is a great method for query optimization that I did not discuss in my original post, however I wanted to highlight that as a great way to optimize queries. Your example of contacts in your phone was very interesting to me in terms of how to find information quickly, and really helped drive the point home about indexing. I was wondering if you had any particular examples from work (assuming that you do not work classified programs) as how indexing sped up query time. Real world examples such as these are often very interesting to me personally.

July 29, 2020 4:21:20 PM EDT

Hey Nick, very interesting to learn about your experience with SQL, and even with your limited experience, you have more experience than me. I have done some work with Oracle before, and I cannot say that I envy your experience in Oracle. I found that it is quite the hassle to deal with all the drivers that come with Oracle, as they are rather large and difficult to manage. Oracle's web documentation also was quite tricky to deal with, so I personally will do my best to avoid it in the future. You mention that your company is moving toward PostgreSQL and MariaDB as alternatives, which I think is a smart move. One other database system that my company seems to be moving toward is DynamoDB, however that is mostly inspired by a company move to AWS and the ease of integrating Dyanmo with other AWS features. Thanks for sharing your experience, I found it very interesting.

## Discussion 10B

July 29, 2020 4:59:55 PM EDT

Hey Juan, I thought you challenges with the database project were interesting. I had not yet gotten to the testing stage of my database project yet, and I am not too excited to hear that was your greatest challenge. Did you write your unit tests for your database within your database tool, or did you use an outside application (python)? Clearly some time will need to be carved out for me this weekend and next week to do some unit testing and verification of my database. Do you have any suggestions for someone that has never done database testing previousy?

July 29, 2020 5:05:50 PM EDT

Hey Adrienne, I thought your example of using drop-downs as business rules for your applications in practice seemed very interesting. Not only would this help to prevent any bad data from getting into your database, it is also probably a good tool for preventing SQL injection attacks. While drop-downs are unlikely to be usable for all of your fields, at least limiting your exposure to a potential threat is always a positive. This also removes the need to ensure that the data users enter is valid, thus having drop-downs are often very helpful to database admins. Thanks for sharing

## Discussion 11A

August 7, 2020 6:15:28 PM EDT

Hey Nick,

One point that I have on your functional based index example with the 5 seccond query is that sometimes it is important to consider the used case of the query. 5 Seconds is generally decently fast, especially with 9 million records. This is especially true if the query is a one-off, or something that needs to be run ~once a day or month for a report. There are relatively few business cases where a complex query needs to be run very often, however if that is the case then the optimization specified is great. I just like to make sure there is a real use case for an optimization before I spend all that time implementing it, otherwise you are not providing value.

Interesting example though, thanks for sharing.

## Discussion 11B

August 7, 2020 6:41:36 PM EDT

Haha, I got a good kick out of your troubles with SQL errors. I experienced much of the same, stakoverflow helped sometimes, but sometimes the error messages are so unhelpful that even stackoverflow cannot save you. Fortunately for me, most of my errors were so silly that it was easy to pick them out. In general I think SQL has pretty simple language and good error messages, however I had the same error message as you for one of my statements, and that was not helpful. hopefully you were able to work through those quickly.