**Database Test 2 Review**

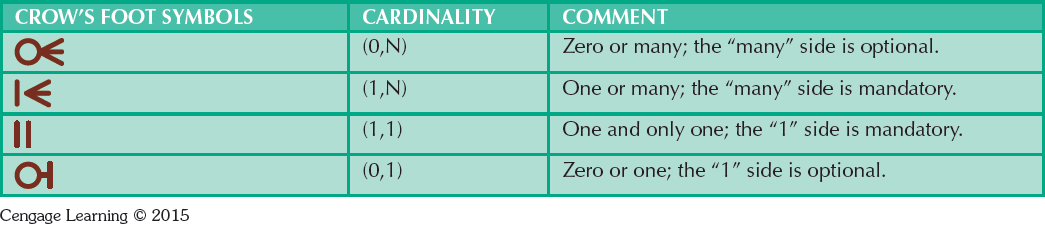
**Test is on Thur, Nov 3rd**

**Overview: Covers Chapter 4, 5, 6 + dapper**

**Big picture: Given a problem description, create an ERD and show corresponding tables.**

**Chapter 4**

* Important Concepts
  + Given a problem description, creating an ERD
    - Pay close attention to the nouns, the kinds of relationships, and whether or not something has sub or super types.
    - Identify business rules based on the descriptions
    - Identify main entities and relationships from the business rules
    - Develop initial ERD
    - Identify attributes and primary keys
    - Revise and rework. (iterative process)
  + You are responsible for Crow’s foot notation only
    - This means no Chen notation on the test.
  + Tradeoff’s associated with storing derived attributes
    - See the derived attribute term below
  + Notation for optional/mandatory participation
    - Optional participation uses a circle instead of a | mark on the relationship lines between the participants.
  + Notation for each of the 4 main cardinalities
    - See the image below
  + Be able to show tables for your ERD
    - To do this, each entity’s listed attributes are made into columns of a table.
  + Implementation options for recursive relationships (fig 4.38)
    - If there is a 1:1 relationship, then just add another attribute points to another entity within the relation. Doing this can cause a lot of nulls though.
    - If it is a many relationship, then another table has to be added, which is essentially a bridge table so that it makes sense for one entity to have a relationship with many of itself.
      * A second table could be added if the relationship needed to be modeled out of it. This would allow for a marriage to be a table, then a bridge table could be formed that holds the marriage num and the employee num for the married employee.
  + Conflicting Goals of DB Design
    - Database design must conform to design standards
    - Need for high processing speed may limit the number and complexity of logically desirable relationships
    - Need for maximum information generation may lead to a loss of clean design structures and high transaction speed
      * Information requirements
* Chapter 4 Terms
  + Composite key
    - An attribute that is made up of more than one attributes.
      * These are used in bridge/associative entities
      * As well as weak entities that would not exist without some other entity to depend on. (dependents in a database.
  + Composite attribute
    - Attribute that can be subdivided to yield additional attributes.
  + Simple attribute
    - An attribute that cannot be subdivided
  + Single valued attribute
    - An attribute that has only a single value (SSN)
  + Multi-valued attribute
    - Attributes that have many values (color options for a car)
      * These are implemented by making each option its own attribute
      * Or by splitting off the options into their own entity
  + Derived attribute
    - An attribute whose value is derived from other attributes using an algorithm
      * These are often stored when the value that has been derived will not change over time. They are not stored if this is not true! If it will change over time, then redundancies will crop up unless it is constantly maintained.
      * Space is not really a huge concern here. It could be that the performance might be negatively impacted though.
  + Relationship terms: Participants/Connectivity/Cardinality
    - Participants are entities that participate in a relationship, which means the two entities are somehow involved with one another.
    - Connectivity describes the relationship classification (1:1, 1:M, N:M)
    - Cardinality expresses the minimum and maximum number of entity occurrences associated with one occurrence of related entity. (0..1, 1..4, etc.)
  + Existence dependence/independence
    - Dependence refers to an entity that can exist in a database only when another entity exists.
    - Independence exists apart from all of its related entities. These are called strong or regular entities.
  + strong entity/weak entity
    - A strong entity is existent independent.
    - A weak entity is existent dependent, and it has a strong relationship with the strong entity it is in participation with. This means that part of its PK is the PK of the strong entity.
      * We do this really only when business rules dictate it. Otherwise, a surrogate key will suffice.
  + Relationship strength (weak/strong)
    - A weak relationship means the related entity does not contain a primary key component of the parent entity.
    - A strong entity is just the opposite. It does contain a component of the parent entity’s primary key.
  + Unary/binary/ternary relationships
    - Unary relationships are those that are maintained within a single entity
    - Binary relationships are when two entities are associated
    - Ternary relationship is when three entities are associated.
      * Bridge tables also fit this paradigm in a sense, but what is shown in the book is a relationship between three entities before a bridge table is introduced.
  + Recursive relationships
    - A recursive relationship exists between occurrences of the same entity set. These are unary relationships that can be implemented as described in the big picture section above.
  + Bridge entity
    - An entity used to allow the implementation of M:N relationships. Often it has a composite primary key made up of the two foreign keys to the related entities.



**Chapter 5 – Advanced Data Modelling**

* Important Concepts
  + How to implement subtype/supertype in tables
    - To implement these in tables, the parent table contains all of the shared information. The discriminator can be used to join the table with the other information that is desired.
    - The child entities contain the attributes in a separate entity for the stuff that is specific to said child. The primary key of the parent is used as a foreign key and the primary key of the child. This is an example where using weak entities really makes a lot of sense.
  + How to implement disjoint/overlapping subtypes
    - A disjoint subtype is implemented by adding a sub\_type discriminator that can be one of any number of options. The value determines which subtype the entity belongs to.
    - Overlapping subtypes contain a bool for each possible sub type. For each one that is true, the parent entity contains attributes that come from the child.
  + What do each of the following desirable key characteristics mean, and why is each important?
    - Non-intelligent
      * This means the key should not contain information that identifies or describes the entity in any way. This is information that someone will likely type in at the keyboard at some time, which means they will probably foul it up.
    - No change over time
      * This means that the primary key selected will always be the same. This is good because if it were to change, all of the dependent attributes would now be determined by a different value, which is not the design.
    - Preferably single-attribute
      * This means that the primary key preferably is something that cannot be another value, such as an assigned ID number of some type. Things like names and colors make bad keys because they could be many values.
    - Preferably numeric
      * If the value is numeric, then the database will be able to handle the incrementing without having to worry about the users input or anything like that. Strings are unable to be incremented, which means they have to be passed in, and they could be misspelled.
    - Security compliant
      * The primary key must not be composed of any attributes that might be considered a security risk of violation. (SSN)
  + List two situations in which composite primary keys are good
    - These are good when a strong relationship is dictated by business rules
    - These are also good when a bridge entity is being used.
  + List situations in which it is a good idea to use a surrogate key
    - It is a great idea to use a surrogate key whenever there is not a natural key.
    - It is a good idea to use a surrogate key whenever there is a composite key (usually).
    - It is a good idea if the natural key is not an integer.
    - It is a good idea to use a surrogate key if all of the attributes contain descriptive information. Good keys are nonsensical.
    - It is a good idea to use a surrogate key if thee primary key would change over time.
  + Describe the preferred way of implementing 1:1 relationships (see Table 5.5)
    - 1 side is mandatory and the other is optional
      * When this is the case, the foreign key goes on the optional side and it must be a mandatory FK
    - Both sides are optional
      * When this is the case, select the FK that causes the fewest nulls, or place the FK in the entity in which the relationship role is played. (like marriage for instance)
    - Both sides are mandatory
      * In this case, do the same thing as in case two, or consider revising your model to ensure that the two entities do not belong together in a single entity.
  + Describe the preferred way of handling time-variant data
    - Create a new entity in a 1:M relationship with the original entity.
    - The new entity contains the new value, date of the change, and other pertinent attributes.
      * An employee will have salary changes over time. These are all stored, so this makes it one to many.
* Chapter 5 Terms
  + Entity subtype/supertype
    - An entity subtype contains unique attributes that do not belong to its parent and are not shared amongst other children (in general).
    - An entity supertype is the generic parent entity that has one or more subtypes.
      * These are part of a specialization hierarchy. The child entity implements an “is a” relationship with the parent.
      * Specialization hierarchy provides the means to support attribute inheritance, which involves much of the below.
  + Partial/Total completeness (plus symbols for these)
    - Partial completeness means that all of the values in the parent entity do not have to appear in a child entity as well.
    - Total completeness means that every value in the parent entity must also be associated with a child entity as well.
  + Disjoint/overlapping constraint
    - The constraint determines whether or not a parent entity might be a part of multiple child entities. Overlapping entities can.
    - Disjoint entities can only be a part of one child entity.
  + **Category symbol**
    - This is the circle with the letter in it that indicates a subtype super type relationship exists. It contains the overlapping and disjoint constraint. It also indicates partial or completeness.
    - This is the thing that we use to discuss what kind of constraint exists between the subtypes.
  + Subtype discriminator
    - The attribute in the supertype that determines to which entity subtype the supertype occurrence is related. The default comparison type for these values is equality.
  + Specialization generalization
    - Specialization refers to a top down process. It identifies lower-level, more specific entity subtypes from a higher-level entity supertype. It is based on grouping unique characteristics and relationships of the subtypes together.
    - Generalization is a bottom up process that identifies a higher-level, more generic entity supertype from lower-level entity subtypes. This is based on the grouping of common characteristics and relationships of the subtypes into the supertype.
  + Natural key (natural identifier)
    - A real-world identifier used to uniquely identify real-world objects.
    - This is good because it is familiar to end users and forms part of their day-to-day vocabulary
  + Surrogate keys
    - A primary key used to simplify the identification of entity instances.
      * These are useful when there is no natural key
      * The selected candidate key has embedded semantic contents or is too long
      * Should use the unique index and not null constraints on these.
  + Time-variant data
    - Data whose values change over time and for which a history of the data changes must be retained. (business rules usually determine this too.
      * The tables these go into are basically like bridge tables.
  + Fan trap problem (and how to fix it)
    - This is a design trap, which is something that occurs when a relationship is improperly or incompletely identified.
      * Represented in a way not consistent with the real world.
    - The fan trap occurs when one entity is in two 1:M relationships to other entities.
      * The great example is associating a division with a team on one side and players on the other. Really, the players should be associated with the team, which is then associated with the division.
    - The fix for a fan trap is to more closely model the real world with your database, which in this case usually means removing one of the 1:M relationships. Rework it so that one of these 1:M’s points the other way.
  + Redundant relationship
    - These occur when there are multiple relationship paths between entities. For example, should a division really know about a bunch of players, or should they know about a bunch of teams that in turn know about a bunch of players? The latter is what should be. If you follow this, then you do not need the first relationship. Having both is redundant. We don’t need two paths between the same entities unless that are expressing a different existing relationship.
      * Model the real world as closely as possible.

**Chapter 6 – Normalization**

* Important Concepts
  + What is “normalization”, and what is its primary objective?
    - Normalization is evaluating and correcting table structures to minimize data redundancies.
    - This process reduces data anomalies and assigns attributes to tables based on determination.
    - The solution to all of the problems in the normalization process is to make another entity.
  + What is the trade-off that must be considered when normalizing tables?
    - The main tradeoff is good logical design structures for query performance. The more entities there are, the more sense a database often makes. However, more tables mean more difficult queries. Additionally, the processing time and speed is often less as well.
  + What characteristics must a table have to satisfy each of the following: 1NF, 2NF, 3NF
    - 1NF
      * There are no repeating groups
      * All attributes are determined by the identified PK uniquely
      * In a table format (this must be attained for an RDB to even exist)
    - 2NF
      * 1NF
      * And there are no partial dependencies.
      * This form can only exist if the PK is composite
    - 3NF
      * 2NF (or 1NF if the key isn’t composite really)
      * And there are no transitive dependencies
        + The anomalies these avoid are insert, delete, and update anomalies. These arise because one version of the data is changed and another is not. Therefore, they end up out of sync with one another.
  + Describe the relationships between 1NF, 2NF, 3NF
    - Each one builds on the last. You have to have 1NF before you can spring up to the others.
  + Given a description of data, create dependency diagrams with all dependencies shown
    - Dependency diagrams require that the PK be underlined
    - That the lines drawn on the top part represent the desirable dependencies
    - That the lines drawn underneath the diagram represent undesirable dependencies that need to be removed in the next iteration of the dependency diagram
    - Foreign keys are not given any special treatment in dependency diagrams.
  + Not responsible for BCNF, 4NF
  + Describe the main motivation for denormalizing tables
    - The main motivation is to meet some performance requirement. Data redundancy is introduced in exchange, but there comes a time where speed really matters.
      * This has drawbacks:
        + Data redundancy
        + Indexing is harder
        + Data updates are less efficient because tables are larger
        + No simple strategies for creating virtual tables (views)
* Chapter 6 Terms
  + 1NF, 2NF, 3NF
    - These terms are defined above.
  + Functional dependence/determination
    - Functional dependence means that a value of A determines one and only one value of attribute B.
    - If the primary key is a composite key, then a subset of the composite key cannot determine any of the attributes or else full functional dependence is not attained.
  + Partial dependency
    - Functional dependence in which the determinant is only part of the primary key.
      * This is assuming one candidate key. It’s very easy to identify.
  + Transitive dependency
    - An attribute functionally depends on another non-key attribute
      * This is a non-candidate key attribute. If it is a candidate key, it can be reasoned that we should just leave the dependency in.
  + Repeating group
    - A group of multiple entries of same type that exist for any single key attribute occurrence.
      * This often occurs in spreadsheets. The “PK” here does not uniquely identify a single entity. This is what has to be fixed before one can enter 1NF.
  + Atomic attribute
    - An atomic attribute is one that cannot be further subdivided.
      * These are the same as simple attributes.
      * They are said to have atomicity.
      * Whether or not this is required is determined by business rules/requirements. They will call for a certain level of granularity that the database is responsible for maintaining.
  + Surrogate key (and Sql Server’s IDENTITY feature)
    - These are used by designers whenever the PK is considered to be unsuitable
    - It is a system defined attribute
    - Created and managed by the DBMS, which is great. Means we don’t have to worry about it in our application code at all.
    - Have a numeric value which is automatically incremented for each new row!
    - In SQL server, you use the identity keyword with a starting number and a number to increment by. It also still requires type information. I also usually add the unique and not null constraint when I use these.
      * P\_ID int identity(1, 1)

**Dapper**

I will give you a sample database and sample code segments (similar to Homework 7). Given this, I will ask you to “fill in the blank” and/or explain the code. If you *understood* what you did for homework 7 (versus simply parroting the example I gave you), you will be prepared for the Dapper test questions. This section, like the remainder of the test, will be closed-book.

For example: Assuming we have classes defined to represent **Employee** and **Job**, fill in the blanks below so that the data types/values are consistent with the remainder of the code.

public static List<Employee> allEmployeesWithJob()

{

using (IDbConnection conn = new SqlConnection(connectionString))

{

string sql = @"select employee.\*, job.job\_code, job.JOB\_DESCRIPTION from employee,

job where employee.job\_code = job.job\_code";

List < Employee > result = conn.Query<\_\_\_emp\_\_\_\_\_\_, \_\_\_\_\_job\_\_\_\_\_\_\_\_, \_\_\_\_emp\_\_\_\_\_>(sql,

*lamda function omitted for brevity*,

splitOn: \_\_\_job\_code\_\_\_\_\_

).AsList();

return result;

}

}

In the above code, employee is listed first and then job because this is the order in which these are mentioned in the query. The final type given is also employee because the code is expecting an employee model to come back in our result. If the lambda function was shown, it would also follow this same ordering convention. Finally, the blank for split on must contain the job\_code attribute because this is the attribute that is used in the where statement.

Misc:

* I will also expect that you can write a sql join statement.
  + You might have to write the join given the Dapper code as well.
  + Joins select attributes from multiple tables where some shared attribute’s value is equal (or otherwise related).
  + Join Examples:

Select Vendor.\*, product.p\_descript

from vendor inner join product on vendor.v\_code = product.v\_code

where vendor.v\_code = @V\_Code

//Used when you pass in a parameter

select product.\*, vendor.v\_name, vendor.v\_contact, vendor.v\_state

from product, vendor

where product.v\_code = vendor.v\_code //when no parameter is needed

* Review all of your relevant homework assignments.
  + Already did this. Spend time practicing instead.