The primary purpose of a web forum is to provide a space to allow users to interact and express their thoughts. Users can submit new threads with text and photos and reply to other threads with text and a photo. Threads are organized by tags and moderated by administrators, who also have the capability to post announcements with video content. To begin constructing a database for such a forum, an ERD can be used to visually map out the entities and their relationships.

Conceptually, it is helpful to consider the entities and relationships in the forum model mainly organized into two entity clusters USERs and POSTs.

Forum members are represented by the USER entity and are uniquely identified though a numerical surrogate key USER_ID. Attributes other than those mentioned in the forum description were added to make the forum more realistic. These include attributes for personal information such as USER_FNAME, USER_LNAME, and USER_DOB, account information such as USER_EMAIL, USER_PASSWORD, and USER_PROFILE_PICT, and descriptive statistics summarizing activity level such as USER_POST_COUNT, USER_STATUS, and USER_LAST_LOGIN. Note that although the forum should constrain each email address to one user, meaning that email addresses could potentially serve as a primary key, a surrogate USER_ID (numerical) was used instead to reduce complexity of the PK that would otherwise include a mix of characters, numbers, and symbols and be harder to work with.

From the Forum description, it is clear that there in addition to USERs, the forum contains some members who are promoted to administrators and moderate the forum. In the ERD, these members are represented by the entity ADMINISTRATOR. ADMINISTRATOR is a subtype of the supertype USER because ADMINISTRATORs have unique administrative privileges in addition to retaining all the attributes of USERs. The completeness constraint between supertype/subtype is only partial because only some USERs are ADMINISTRATORs. It is not relevant to specify disjoint /overlapping constraints because there is only one USER subtype in the model. However, if there were to be additional attributes unique to general users, this model would need to be adjusted to include other USER subtypes.

Alternatively, it may have been tempting to just identify administrators by simply adding the attributes of administrators to the USER entity. However, because only a small subset of USERs are ADMINISTRATORs and because ADMINISTRATORs have unique characteristics, it would have not be appropriate because that would introduce a lot of nulls to the database. Likewise, it would also not be appropriate to create a completely unrelated entity for ADMINISTRATORs because they still need to have all the characteristics of USERs. Therefore, the subtype/supertype structure is preferred.

The main way USERs can interact is through submitting posts, the types of which are organized in a supertype/subtype specialization hierarchy to reduce redundancy. The POST entity uses a numerical POST_ID as its primary key, and is a

supertype of the complete disjoint THREAD, REPLY, and ANNOUNCEMENT subtypes. A POST must be either a THREAD, REPLY, or ANNOUNCEMENT, which is identified by the subtype discriminator POST_TYPE.

and share the general attributes included in POST. USER_ID is a FK attribute linking a POST subtype to the USER or ADMINISTRATOR who posted it. While THREAD, REPLY, and ANNOUNCEMENT subtypes are weak entities in relation to POST and although each POST must be created by a single USER, POST has a non-identifying/weak relationship with USERs. The POST_ID is assigned a value corresponding to the order they are created and is not derived from the USER_ID (to be implemented at the programming level) . Because POST_ID alone is enough to uniquely identify each post , it is not necessary to include USER_ID as a PK. The benefits of such a design means that if for some reason a USER_ID is deleted, the POST occurrences related to that USER_ID will remain and the content will be preserved (but perhaps with the USER_ID field removed and replaced with a default value depending on the system creator's implementation).

Notice that despite the link between POST_ID and USER_ID, there is no relationship drawn between the POST and USER entities. That is to enforce the constraint that only

ADMINISTRATORs have the privilege of posting ANNOUNCEMENTs. Although it may appear cluttered, it was not appropriate to draw a relationship directly between USER and POST because that would allow ANNOUNCEMENTs to inherit a relationship to USER. Instead, an optional, non-identifying, weak 1:M relationship was drawn between USER and REPLY, USER and THREAD, and ADMINISTRATOR and ANNOUNCEMENT.

Furthermore, notice that the PK of REPLY is a composite key composed of the POST_ID and REPLY_NUM. Unlike THREADs and ANNOUNCEMENTs, new occurrences of REPLY are not assigned a unique POST_ID. Instead, the PK POST_ID in the REPLY refers to the THREAD or ANNOUNCEMENT to which the REPLY is replying. For each THREAD or ANNOUNCEMENT, REPLY_NUM starts at 1 for the first REPLY and increases +1 for each subsequent REPLY (to be implemented at the programming level). Such a composite key allows REPLYs to be uniquely identified while preserving the order in which REPLYs are submitted and maintaining the reference to the post it is responding to. To preserve integrity and avoid potential nulls in the PK of other entities in the model when manipulating the data, the system creator should set the default REPLY_NUM to 0 for any POST_ID that doesn't have an assigned REPLY_NUM. This wasn't done automatically in the database to save memory.

Alternatively, another PK option for REPLY would have been to assign each REPLY occurrence an unique POST_ID value; however, this would make the list of POST_IDs significantly larger and obscure the relationship between REPLY and THREAD (by facilitating REPLYs to be queried out of context), which

is usually not desirable. It would be possible to avoid the first issue if a new attribute REPLY_ID was created, however, unless REPLY_ID is derived from POST_ID, it would also suffer from the second issue.

Although touched on before but not explicitly stated, each THREAD and ANNOUNCEMENT has an optional 1:M relationship with REPLY, which has in turn has a mandatory 1:1 relationship to THREAD and ANNOUNCEMENT. THREAD or ANNOUNCEMENT do not need to have a REPLY to exist, but a REPLY must reference a THREAD or ANNOUNCEMENT to exist.

The reason why THREAD and REPLY could not appear in the same table is the constraint on the multimedia stated in the forum description. A THREAD has a 1:M relationship to PICTURE while REPLY has a 1:1 relationship to PICTURE. Similarly, the reason why THREAD and ANNOUNCEMENT entities could not be combined within the same table is that ANNOUNCEMENT have a 1:1 relationship with VIDEO while a THREAD is not permitted to have any videos. Although these requirements could be left solely for the forum programmer to implement, it adds an extra level of system integrity at the database level to include it in the ERD. Additionally, the benefit of this design also allows the system creator to implement further constraints on the content of the different types of POSTs, such as allocating a larger char limit for content in ANNOUNCEMENTS and THREADs compared to REPLYs.

To further elaborate on PICTURE mentioned above, PICTURE is an entity that associates an image in a post to the external server location where the actual image is stored. Because each PICTURE can only appear in either one thread or in one reply (taken to mean that all PICTURE addresses are unique), it is practical to use the surrogate PK PICTURE_ID, which is a numerical value assigned to each picture as it is posted. Because the PK of PICTURE does not include the PK of THREAD or REPLY, the relationship is weak and non-identifying, which is represented by the dotted line in the ERD.

The relation to the post in which it appears is made through the setting the PK of its parent post as a FK . Since the PK of REPLY is (POST_ID, REPLY_NUM), but the PK of ANNOUNCEMENT and THREAD is POST_ID, the FK must take on the format POST_ID, REPLY_NUM). Therefore, REPLY_NUM should be given a default value of 0 if none is given as explained above. (Note that the table for POST includes the derived attribute POST_HAS_PICTURE to aid in search and retrieval speed for the end user.) The connection to the external server is made with the attribute PICTURE_ADDRESS, which was not labeled a FK because the picture hosting server is not an entity represented in the ERD. Although PICTURE_ADDRESS is a unique identifier, it would not be ideal for use as a primary key because addresses can be complex and cumbersome to use as an index.

Alternatively, If PICTURE was an attribute, it would not be possible to depict the previously mentioned constraints in the ERD, and the forum would have to rely completely on the programmers implementation of the business rules.

Moreover, since not every posting in the forum is going to have a picture and some will have multiple, attempting to store pictures alongside other post attributes would fill the database with nulls. To go a step further, if images were not distinguished from text content and the image URL was just grouped with text content, searching through the content of the forum would be slow and possibly unreliable and the length of posts with images may be significantly limited by the length of a URL. For practical purposes, the system creator should set a limit to the maximum number of PICTUREs a THREAD can contain to prevent a single THREAD from taking too much space.

In addition to PICTUREs, the content of some posts, specifically ANNOUNCEMENTs, can include a video. VIDEO is a component entity of ANNOUNCEMENT and is identified by a surrogate PK VIDEO_ID (indexed in the same way as PICTURE). Each VIDEO must be contained within an ANNOUNCEMENT and is related through the FK POST_ID. (Since POST_ID alone doesn't distinguish the type of POST, it is up to the programmer to constrain this requirement at the end user level.) Each ANNOUNCEMENT can have at most one video. The relationship of ANNOUNCEMENT to VIDEO is optional 1:1 while the relationship of VIDEO to ANNOUNCEMENT is mandatory 1:1. However, the relationship is weak/non-identifying because the PK of the parent doesn't appear in the PK of the child. The video can still exist in the external server without being associated with an ANNOUNCEMENT.

Alternatively, since VIDEO only requires one unique attribute and is associated with one entity, it would have been easy to just add a column attribute to ANNOUNCEMENT to store the VIDEO address. However, assuming there will be far fewer videos than ANNOUNCEMENTS, it would be more efficient and avoid nulls if VIDEO was placed in a separate table.

The TAG entity stores keywords that help to search and organize THREADs and ANNOUNCEMENTs by topic or subject matter. TAGs are uniquely identified using TAG_ID, which is a numerical surrogate PK attribute that is derived from the TAG_LITERAL attribute (the literal keyword for the tag). Note that TAG_ID was chosen as the PK rather than TAG_LITERAL to avoid having an index with semantic meaning and mixed datatypes/variable sizes. Moreover, adding this attribute supports the programmer in standardizing TAGs so a post tagged with TAG_LITERAL value "hw1" and another with "HW_1" can both be retrieved on a search for "hw1." Additionally, storing the TAG_LITERAL will allow the end-user-entered TAG to display, which may be helpful in identifying and quickly correct errors in standardization.

The person who creates each THREAD or ANNOUNCEMENT can include multiple tags and each tag can refer to multiple posts. Since this M:N relationship cannot be represented in an ERD, a bridge entity TAGGED was used to split up the M:N relationship into two 1:M relationships. A THREAD or ANNOUNCEMENT can have multiple TAGGED references and a TAG can be used in multiple TAGGED references. There can be no TAGGED occurrence without a TAG or a THREAD/ANNOUNCEMENT, so TAGGED is completely

existence dependent on its parents. Because TAGGED uses the PK of its parents as a PK and is existence dependent, TAGGED is a weak entity.

The LIKE entity is essentially a bridge table between USER and POST that records who has liked what. Each USER can LIKE many POSTs (optional 1:M relationship), but a POST can only count one like from a specific USER (mandatory 1:1 relationship). Each POST can have many LIKEs (optional 1:M relationship), but a single USER can only like a specific POST once (mandatory 1:1 relationship ). A LIKE cannot exist without both a USER and a POST. LIKEs are identified using a composite PK composed of the PKs of USER and POST (POST_ID, USER_ID, and REPLY_NUM). Therefore, LIKE is a weak entity with strong, identifying relationships to USER and POST.

There are two levels of LIKE ("like" and "superlike"), and it is assumed that they are mutually exclusive. It is not possible to simultaneously like and really like something. The levels of LIKE are stored as an attribute of LIKE rather than as subtype entities. This is preferable because it is efficient without being destructive. "Like" and "superlike" only differ in one characteristic, and the mutual exclusivity means that such a design will not increase redundancy nor introduce nulls. For the sake of retrieval speeds, the derived attributes POST_LIKE_COUNT and POST_SUPERLIKE_COUNT are included in the table for POST.