Database Final Project
Lucas Grant, Shivani Murali, David Earley

Stack Overflow Project Written Report

E-R Model Part 1

In order to create the E-R diagram, we looked at the seven requirements, and tried to construct the diagram by satisfying all seven requirements. Initially, we had a relation for User, Post, Comment, Favorites, Votes, Tags, Questions, and Answers. After we had designed our rough model, we considered whether the Question and Answer entities should be a hierarchy with Post, instead of having relation sets to it. We were concerned that it would be a hassle to make the Question and Answer entities separate because they had many fields that were identical. Unfortunately, we didn't know how we would represent the hierarchy structure in MySQL. After some deliberation, we decided that if we address some other issues we might not have to worry about the Question and Answer relations, so instead we moved on to the relations Comment and Vote. Since Vote had to be applied to both Question and Answer, we didn't know if we should make it have a relation to Post instead of two relations to Question and Answer, and we ran into the same problem with Comment. We thought we would need to have a Question Comment and an Answer Comment, which seemed extremely inefficient but with our current model we did not know any other way to capture that requirement. After some thought, we decided to create a single comment entity and have two different relationship sets to both the Question and Answer entities. Then we decided to have tag coming off of the question relation, and it also had a connection with user since only the user can post tags. We thought that it was a rough diagram to say the least, so we used this as our first draft.

Relation Model Part 1

Once we had our first draft of our E-R diagram, we needed to define a relation for each of our eight entity sets. Then we need to define relations for each of the relationship sets in our E-R diagram. After that step, we had 19 separate relations. We then modified our keys for each of the relations that we created in the last step to capture mapping constraints. All of our relations didn't require any changes, so then we eliminated identical relations, which did not get rid of any extra relation. Then we had to merge relations with the same primary keys to capture participation constraints. This took our original 19 relations, down to just 9. We had Favorite(F_Id, user_id, q_id), Answer(A_id, Post_id, q_id, accepted_id), Comment(c_id, q_id, comment_creation_date, body), vote(v_id, post_id, up, down), Post(p_id, user_id, p_creation_date, title, body), question(q_id, post_id), User(user_id, location, account_creation_date, display name), tag(name), and isTagged(Q_id, name). We then went about connecting the foreign keys of all of these relations, and then created the Tables inside of MySQL.

Data Import part 1

We had our database tables set up, the file with the data was downloaded and waiting to be imported into the database, but had a lot of trouble importing the data. We had to get rid of much of the data since our database wasn't properly set up for it. Due to this setback, we needed to rethink our E-R diagram and our Relational model. This obstacle made us realize that we

could create a better ER model and relational model in order to better capture the seven requirements and design our database.

E-R model part 2

After we decided to redo our E-R model based on the data we were given for the seven tables, we were able to design a final model that best captured all seven requirements. The second time we created the model we had less trouble capturing the requirements. First we decided that we should not have separate Question and Answer entities. We decided to just create a single Posts entity that contained a p_type property that would determine if the post was a question or an answer. If p_type was 1 then it would be a question post, else if was a 2 it would be an answer post. After that we also created a parent_id that should only be used if the post type is an answer post because answers are connect to a question and to determine the question post that the answer is connected to we use this property. If the post type is a question post, this property would just be null. We also created an accepted answer post id property with posts to to capture that requirement but this property should be null if its an answer post. Then, we took the big relations, such as Posts, Users, and Comments, and determined how each of the relations should be related to each other. After we determined those relations, we needed to handle the votes and favorites but we had some trouble with that. After some thought, we realized that they could make them into a relationship set between user and post and we added in their attributes. We could also have done that with Comment, but we thought it was better to have it as an entity that had a relation to Post alone. Our model still is not completely perfect because it cannot capture the constraint that a user can only vote or favorite a post once. Even though our model does not do this we believe it captures all other requirements well. Finally, we felt confident that our E-R model was well built and captured the necessary constraints as best we could.

Relational Model Part 2

After we remade our E-R model, we made a new Relational Model that was much simpler, and that we liked much better. First we defined relation for each of the entities and we had for relations (user, post, comment, and tag). Then we defined relations for each relationship set. WE had four relationship sets (user, post, comment, and tag). In order to define these relations, we used the primary keys of the two tables it was connecting and we also added attributes of the relationship set as a property. After we did this, we modified keys to capture mapping constraints. Most of our relations were many-to-many so we did not need to make any changes to those relations. We had one many-to-one relation, so we made the primary key of the relation to be the key of the entity set on the many side. This allowed us to go from Holds (p_id, c_id) to Holds(c_id, p_id). The next step we had to consider was eliminating identical relations but since we did not have an identical relation, we did not have to do anything for this step. Our last step in creating a relation model is to merge relations to capture participation constraints. For total participation we have to merge relations and define foreign keys as not null. We then had to merge together Comment(c_id, c_date, text) and Holds(c_id, p_id) this then resulted in Comment(c_id, p_id, c_date, text). We then final created our relation model and drew lines to show where the foreign keys come from.

Database Final Project
Lucas Grant, Shivani Murali, David Earley

Data Import part 2

This time around, since we built the database around the information provided, the data was imported almost perfectly. A few minor things we had to change were the names of all the data fields to fit with what we defined the tables as. We had to delete the old parameters and reload the correct ones in. We also had to change the default type for some of the parameters. For example, originally we had varchar(60) for the body but after we imported the data we realized that some of the content was being cut off so we had to increase the size of the type in order for all of the content to display properly . Other than that, the data fit into the tables perfectly, and we had a working database to write the queries on.

Query Optimization

To optimize our first queries, we tried out many different solutions. We tried adding in different indexes to try and make the query run faster. Before we added any indexes, the query executed in 1.64 seconds. Using the explain, we saw that it was using a full table scan when it was searching through the posts to see whether it was a question or answer being asked. We thought a good solution would be to use an index over the post type (question or answer) to make it faster but unfortunately this actually only slowed the process. After trying out a few indexes over the tag name, the username, and parent_id, we found that they did not make the query faster either. By finally adding an index over post title, we found it made it a little faster and we could run the query in 1.32 seconds. We also tried adding in straight_joins which did not help the execution time.

Next we tried to optimize the second query. Originally, we had the condition where we looked up where the tag name equals ‘java’ be placed after we made the joins and it came out to 1.28 seconds. We made it faster by moving the selection down before we joined the tables so that it would have to query through less results. The final execution time came out to be 1.00 seconds.

We then ran the third query and originally it ran 1.46 seconds. We thought that making an index on post date would make the query run faster, but it had only remained the same. We did not keep that index in there.

After that we ran the fourth query and it ran at 1.62 seconds. The index on title also did not make this query faster. We tried to use striaght_joins but this did not help. We also tried counting votes using a different field of vote but that made it slower. We also tried to use an index on vote type and that made it a lot slower.

This fifth query originally took 1.18 seconds to execute. The index on title we had for previous queries made it slower so we took that out. We tried using straight_joins in place of our joins and they did not make out query run faster. We tried joining the comments table before joining with the favorites and answer sub-query and it made only slightly faster. In our original query we already selected a question with a specific p_id before we joined with other sub-queries so that was also optimal compared to filtering after joining with all the tables.

Database Final Project
Lucas Grant, Shivani Murali, David Earley

Overall, even though we tried to add indexes to properties based on our where condition, we could not get the query to execute faster. Selecting before joining the sub-queries helped the execution time. We also had to make sure to clear the cache because if we didn't clear the cache the query would seem as though it is running faster because it is being stored in the cache.