CoVal 2.0

LMNO

Jonathon Leight
Trevin Maerten
Noel Nacion
John O'Conner

RIT Office Co-op and Career Services

Jim Bondi

Faculty Coach

Daniel Krutz

Project Overview

The RIT Co-op Evaluation system is a system by which students and employers can submit evaluations about their mutual co-op experience. The system is accessed via a browser. Users must authenticate themselves before they will be given access to the system. Once authenticated, the users can fill out and submit any evaluations that they are authorized to fill out. The evaluations can be saved and edited at a later date for convenience. The evaluation system also allows certain authorized users to review the submissions and preform analysis on collections of submissions. Notifications can be sent by the system to its users to remind them to fill out the evaluations when the deadline for submissions approaches.

There are several user classes that would be using the RIT co-op evaluation system. First there are the students who go on co-op. They need to use the system in order to submit their own evaluation of their experiences on their co-op. This submission is a necessary part of their grade. Secondly there are the employers of the co-oping students. They too have to complete and submit an evaluation of the student. Thirdly are the RIT faculty which are given authorization to access co-op submissions for students and employers. Access to the submissions is restricted to those within the same college as the student. The faculty can also manually send out notifications to either students or employers to remind them to complete the evaluation. The ability to aggregate the data is also given to the RIT faculty. Lastly there are the system administrators which are given extra privileges in order to maintain the system.

The original RIT Co-op Evaluation system was built using Java and Oracle. The current CES was developed almost 10 years ago. Over time, development teams have added onto the initial functionality as the user's needs changed. The current system is still built upon the same technologies that were used in its original design. The new proposed system will be designed from scratch with all of the existing features in mind so that the system is designed as a whole and not piece by piece.

The new version of the CES, known here as CoVal2, will be a complete re-design and re-implementation of the CES making use of newer, better technologies including C# and MVC3 with MSSQL. The implementation will use standard tools and standard libraries for easier maintenance. An emphasis will be placed on performance and usability for the design and implementation of CoVal2.

The primary stakeholders for this application will be:

- Rochester Institute of Technology
- Office of Cooperative Education and Career Services (OCECS)
- RIT Students
- Co-op Employers
- RIT Faculty

Basic Requirements

CoVal2 will provide the ability for evaluations to be submitted for approval. The evaluations must be approved by a user with the correct permissions. The specific evaluations given can be customized to suit the needs of the department requesting the evaluation. A notification system will be included with the system in order to notify users of any unsubmitted work evaluations. Notifications will be sent by email to whatever address is associated with the user within the system. The system can also create reports based on the submitted work reports. Past submitted work evaluations can be searched for and reviewed as needed.

The system has 4 main user groups, and the functionality is based on the user accessing the system at the current time. They serve as the main inputs to the system. Each user has restricted permissions and they are only able to use functions relevant to their needs. The user groups are defined below.

Student

The student is one of the key users for this system. As part of the requirements for completing a co-op a student must also submit a student work report for approval. This evaluation will be submitted online through CoVal2. Once an evaluation has been submitted, it will be stored on the server and the student will be able to look at the evaluation at any time.
Since most students do not need to use the system more than once per co-op term, our main concerns for students are usability and performance. High usability is important to us because students do not spend a lot of time on the system, so they need to be able to quickly figure out what they are doing so that they do not need to spend any more time on the system than is necessary. Performance is also important for the same reason.

Employer

The employer is another of the key users of the system. As with the student, the employer must also fill out and submit a work evaluation form for approval. While the

employer isn't directly given a grade as part of the evaluation, the evaluation form is still necessary in order for the associated student to receive a grade.
Like with the student, the employer's main concerns with CoVal2 will be usability and performance. The employer will only need to use the system when completing evaluations and looking up previous evaluations. As such, our concerns for employers are the same as for students.

Evaluator

The evaluator is the user in the system which will evaluate the submissions. After a student has filled in his evaluation, the student is eligible to receive a grade for their co-op. The evaluators will have access to both the student and employer evaluations relating to their department. The evaluators will also be able to send out notifications to both students and employers reminding them to complete the work evaluations. Evaluators also will have the ability to search for specific work evaluations, both ones that have been evaluated and ones that have not.
The evaluator's needs for the system would also be centered on performance. They would be using more of the system's functionality than regular users and thus would necessitate that the system is quick and responsive. Usability isn't as high as a requirement since evaluators use the system more often than students and employers and will not be as likely to forget how to use the system in between uses, but it is still a necessity as they would be working with the system frequently.

Administrator

The administrator has all the abilities of a regular evaluator as well as specific administrator abilities. The specific administrator abilities center on user management. The administrators would be able to authorize users as students, employers, evaluators or administrators as necessary.
The needs for the administrator would be similar as the evaluator but with a higher emphasis on accessibility. Since the administrators are in charge of maintaining the system it should be available as often as necessary in order to perform the necessary actions.

The evaluations themselves have multiple states. Evaluations, both employer and student, will start out as Pending. In the pending state, the evaluation appears in the system, but cannot be filled out yet.
Three weeks from the end of a term, all evaluations will be changed from Pending to Open. An evaluation in the Open state can be filled out by the employer or student that it is assigned to. An open evaluation can be changed to the Saved, Submitted, Manually Completed, or Archived state.
If the user saves the evaluation, the evaluation is changed to the Saved state. In this state, the user can open the evaluation and continue filling out answers, and can continue saving the evaluation until they are finished with the evaluation. The Saved state will be referred to as "In Progress – Saved" in the user interface. The evaluation can change to the Submitted, Manually Completed, or Archived state from the Saved state.

The Submitted state is reached by submitting the form from either the Open state or the Saved state. In this state, the evaluation cannot be changed by any class of user. At this point, the only change that can be made to the evaluation is either an evaluation approval change. If the evaluation is approved, nothing else will happen. If the evaluation is rejected, the evaluation will go back to the Saved state.
There are two other states that an evaluation can end up in: Manually Completed and Archived. If the evaluation is completed in some way other than the normal process, an evaluation can be changed to the Manually Completed state. If the evaluation will never be completed, and the user does not want to receive messages telling them to fill out the evaluation, the evaluation can be marked as Archived. The Archived state was known as the Past Pending state in the previous version of the CES.

The main focuses going into the redesign were performance, availability, security, usability, and portability.

Performance

By utilizing the third normal form and removing any redundant data, there is less data to parse through when querying the database for information. The stored procedures used by CoVal2 aid in data retrieval and decrease the bandwidth used during querying by limiting the amount of communication between the client and the server. Using stored procedures and the database to their full potential will help increase the performance of the system.

Availability

The system is designed to continue running with minimal maintenance during most of the time. The system only requires high availability during the first few weeks of a new period, and the system will provide ample availability during this time. A focus on error detection and prevention will help to make sure that the application is always available when it needs to be.

Security

All access is granted based on a user's type. An unregistered user cannot access the system or its data in any way. User passwords are stored only as a hash of the password in the database to ensure no one besides the user knows their password. Session information is stored as a cookie on the user's machines and cannot be accessed by anyone but that user during that session. Particular attention will be paid to user roles in the system, and all controllers and actions will be secured in order to prevent unauthorized access.

Usability

CoVal2 has been designed with a number of usability features, specifically in the user interface, but there are a number of features that also need to be accounted for in the backend of the system. CoVal2 will include an auto-saving feature that will save user's input automatically when filling out forms. This is taken into consideration in actions that can act on either the full set of data, or just a partial set of data.

Portability

CoVal2 is designed to run on all major, up-to-date browsers including but not limited to Mozilla Firefox, Microsoft Internet Explorer, and Google Chrome. This means that the user

interface will be written in standards compliant HTML and CSS, using widely supported JavaScript libraries, and very little custom JavaScript that must be maintained by the OCECS.

Constraints

There are technical constraints that CoVal 2.0 had to be designed around. The environment in which the system will exist on is one of the most significant constraints. CoVal 2.0 will be hosted on a single machine in a Windows environment. Therefore all the technologies that CoVal 2.0 will be built off of must be compatible with a Windows hosting environment. In addition the technologies have to be compatible with each other so that the system can effectively be hosted.

CoVal2 will be designed to work on the system that the OCECS currently uses for both the CES and their other current applications. This includes the following components:

- Windows Server 2008 R2
- Microsoft SQL Server 2008
- C# .NET
- ASP .NET MVC3 Framework

Other libraries on the NuGet[1] gallery, as well as open source libraries may be used, so long as the libraries do not require any of the following:

- Participation in a community forum
- Registration of company information
- Licensing fees
- Sharing source code and/or binaries

The following libraries are already in use in other OCECS applications, and should be considered requirements if their functionality is desired in CoVal2:

- Data Annotation Extensions for more data annotations
- ELMAH for error handling and logging
- Foolproof Validation for more data annotations and validators
- jQuery for client side JavaScript
- Ninject for dependency injection
- NUnit for unit testing

Finally, the source code and any other artifacts that need to be version controlled will be stored in the OCECS Mercurial repository.

Development Process

LMNO used the spiral methodology. This will allowed us to check with the project sponsor to ensure correct interpretation of the requirements and a steady development process. It also

[1] http://www.nuget.org/

allowed the project sponsor to gauge progress on the overall project. The length of each spiral was two weeks and consisted of all of the deliverables that were required for those two weeks.

Project Schedule: Planned and Actual

The original plan for development was to finish the requirements, architecture, and underlying database implementation in the first quarter, and then to finish the rest of the work the second quarter.

We were successful in finishing all of the work we expected to the first quarter. We had a slow start the second quarter due to our unfamiliarity with the development tools that we were required to use, but progress was being made. By the middle of the quarter, we had about 50% of the core functionality completed, but the product was not up to the team's standards. In order to fix this problem, we took a break from adding more functionality to go back and fix the problems, and bring the product up to our standards.

This re-implementation cost us valuable time for getting more functionality complete, but it ended up being a necessary step in the process as it made the customer happy to be receiving a piece of software that was up to their standards. We then renegotiated the scope of the project to account for what we would realistically get done during the rest of the quarter.

The new scope was agreed upon, and the team was able to complete all of the work in the agreed upon scope by the end of the quarter.

System Design

Database Design (Part 1)

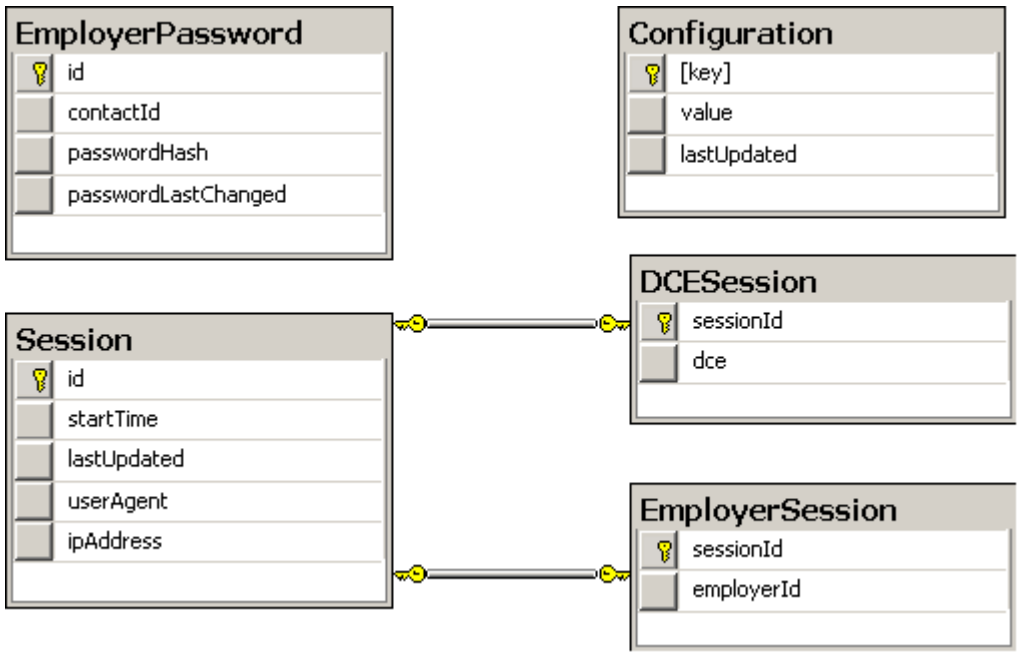

Figure 1: Session, Configuration, and Employer Password tables.

Element Catalog

Other than the session tables, these tables are not really connected in any way. The separation of these tables was just to make the design easier to understand.

1. EmployerPassword
   Contains all information to save an employer's password.

2. Configuration
   Contains a configuration value and the time the configuration was last updated.

3. Session
   Contains information about a session, including the user's IP address, the time they started using the system, the time their session was last updated, and their user agent. Two types of sessions must be maintained: DCE sessions and Employer sessions. These are identified by corresponding entries in the DCESession and EmployerSession tables.

4. DCESession
   Links the DCE of an RIT user to their session.

5. EmployerSession
   Links the employer's ID to their session.

Context

This database diagram is part of the CoVal2 database. Since MSSQL does not allow for cross-database foreign keys, there is no foreign keys to the OCECS database, but there is, in spirit, a link between employerId, contactId, and the Contacts.id field in the OCECS database.

Architecture Background

The EmployerPassword table was designed with usability in mind. It allows for an employer to have their own custom password, and one single account, rather than having multiple accounts and passwords for each evaluation that they need to fill out.

The Session tables were designed with usability in mind as they allow students and employers to log in to the system and maintain their session rather than being kicked out of the system as soon as they close their browser or their CoVal2 tab.

Database Design (Part 2)

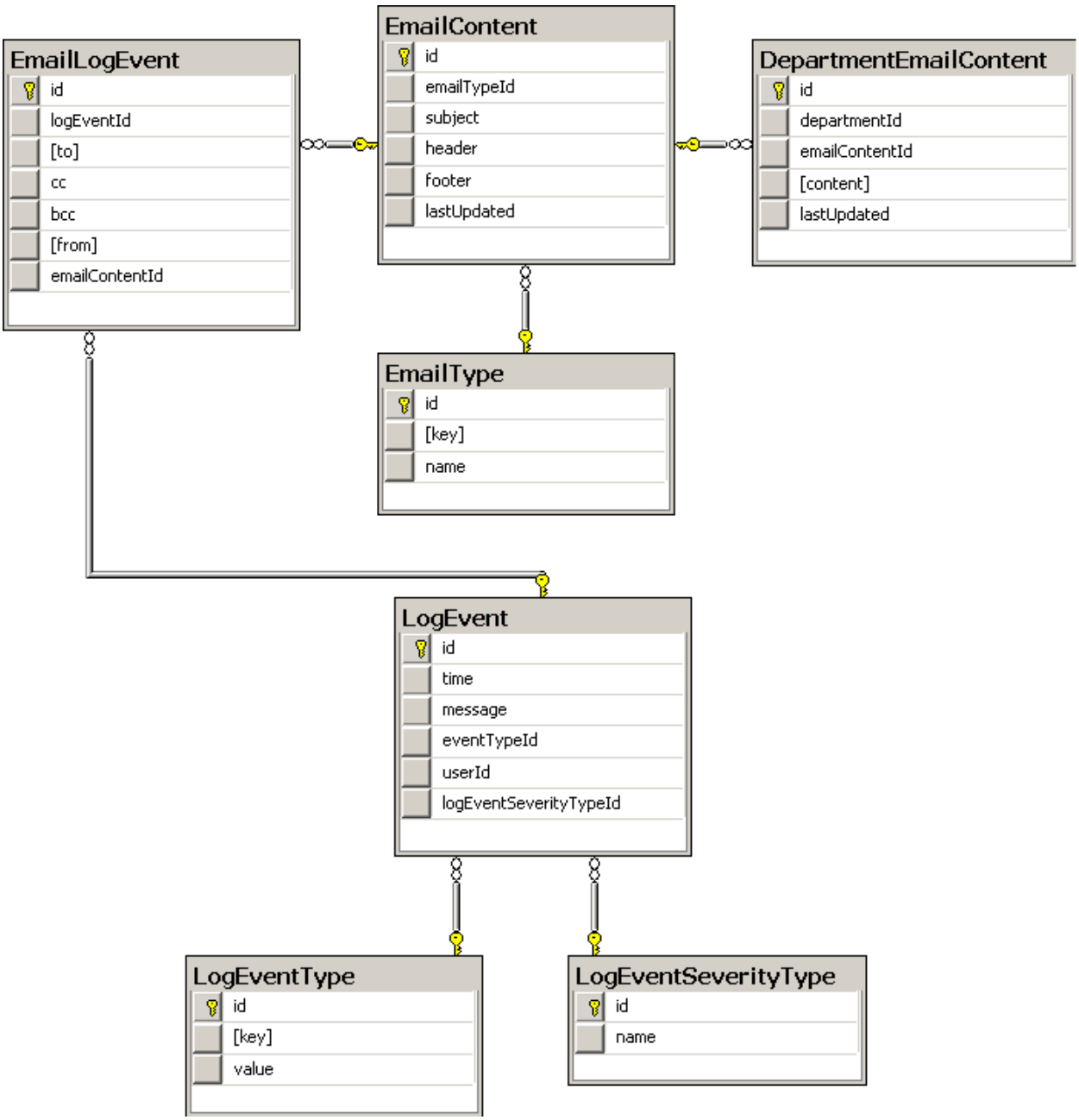

Figure 2: Event Log and Email Notification tables

Element Catalog

The tables in this part of the database design all revolve around the event log and the email notifications that will be sent out.

1. LogEvent

   A log entry representing an event in the system including a date and a time, a message explaining the event, which user caused the event, the type, and the severity of the event.

2. LogEventType
   This table contains a list of all of the different event types that can occur in the system.

3. LogEventSeverity
   This table contains a list of all of the different severity levels a log entry can be.

4. EmailLogEvent
   Information about an email, including the subject line, sender, receivers, and a link to the content.

5. EmailContent
   The content of an email, including the subject line, header, footer, and the time it was last updated.

6. EmailType
   The type of an email.

7. DepartmentEmailContent
   Special department specific information in an email.

Context

The tables in this diagram are all part of the CoVal2 database. The tables, except for LogEvent, are all self-contained and do not have foreign keys to any other system. The userId field of the LogEvent table will link to the actual user that caused the event.

Architecture Background

Given that the system must be both performant and available, it was decided that there should be an event log in the system. This allows for an easier time figuring out if there are errors with the system or if something is going on to make the system less performant.

Given that the system is supposed to be usable, the email related tables have been designed so that departments can change their own email notifications, and that changes are kept separate from the overall body of the email messages. This allows for modified email messages to be easily detected, and allows for modification of the overall message without breaking department specific modifications and vice versa.

Database Design (Part 3)

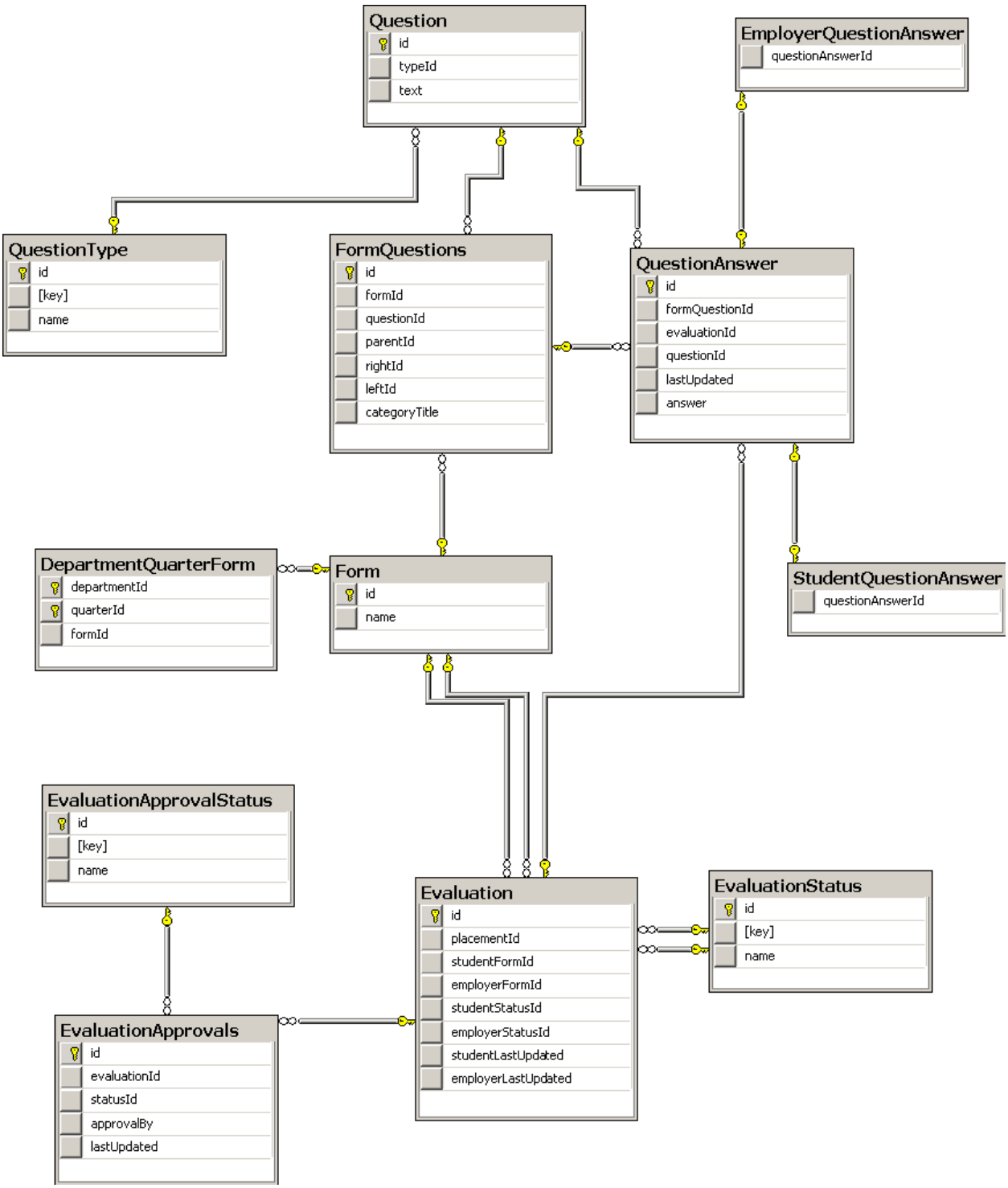

Figure 3: Form related tables

Element Catalog

The tables in this section of the database deal with the Form, Questions, and Evaluations.

1. Evaluation
   Contains references to the placement, student work report, employer evaluation, evaluation status, student status, and the times the student and employer were last updated.

2. EvaluationStatus
   The status of an evaluation. This is linked twice as there are statuses for both the student and employer evaluation.

3. EvaluationApprovals
   Contains a link to an evaluation and it's status. Contains who approved the evaluation.

4. EvaluationApprovalStatus
   Contains the status of an evaluation and a link to an evaluation.

5. Form
   Contains the name of the form.

6. DepartmentQuarterForm
   Specifies which form should be used for which department in which quarter.

7. FormQuestions
   Contains links to the form, the surrounding questions, the question itself, and the question category.

8. Question
   The question and the question's type.

9. QuestionType
   The type of a question. This includes, but is not limited to: text, number, date, Lykart, and double Lykart.

10. QuestionAnswer
    Contains a serialized answer to a question and the time it was last updated. Also contains links to the question, the evaluation, and the FormQuestions join table.

Context

The tables in this diagram are all part of the CoVal2 database. The only links to external tables are the approvedBy column in EvaluationApprovals, and the placemendId in the Evaluation table.

Architecture Background

These tables were designed mostly with performance in mind. They were designed to third normal form to try and reduce the duplication of data, and indices were added where they would increase performance during queries.

Binary serialization for the answers was chosen due to the large number of different possible answer types that could exist in the system. Rather than making a large number of tables that linked together in complex ways, a single table with a binary column was chosen to represent the answers. Given that the system needs to be performant, CLR stored procedures have been considered to allow the database to process the binary data without having to query all of the data, send it to an application, deserialize everything, and then finally be able to work with the data.

Since the system still needs to be usable, CLR stored procedures will be made that allow the database to be queried directly without the use of an application capable of deserializing the binary data. This will allow for the most flexibility, with little to no performance cost.

Other Designs

While there were other iterations of this design, the differences mostly revolved around the storing of the different question and answer types in the database. The final decision was to serialize the information that would not fit into a single table, so that we could store it in a single, common table. This ended up working out well for us, and the speed at which we could query the data in the database was much faster than the previous CES.

Process and Product Metrics

LMNO used two metrics to help track progress: the amount of time spent by each team member, and the number of bugs found in the system.

The first metric, the amount of time spent by each team member, was a little less useful than the second one. It was, however, able to tell us that the scope for the project was a little too large. Given that everyone was spending about 15 hours per week on the project, yet we were not getting as much done as we expected, we ended up using this to help us renegotiate the scope of the project.

The number of bugs found was more helpful since it showed us that we needed to refactor the project half way through. The number of bugs that we found in the system increased steadily until the refactor. After the refactor, there were fewer bugs left in the system, and the number of bugs found, while still increasing, was increasing at a slower rate.

Product State at Time of Delivery

The project, at time of delivery, has all of the agreed upon features implemented and working. Some tweaking will need to be done to make some of them work exactly as the customer wants them to work. The major goal of the project was to get the core functionality working, as well as documenting all of the things that would make the application more usable. This will help the OCECS finish the project, as well as giving them a set of requirements and a possible architecture that they can use to make a new evaluation system using new technologies in the future.

Project Reflection

The team worked together well during the project, and was able to complete the agreed upon functionality that was required by the customer.

The team learned that inexperience with the development tools will hinder the progress of a project immensely. We originally thought that we would be able to get a lot more done than we ended up getting done. Our unfamiliarity with the technologies lead us to a renegotiation of the scope of the project half way through the second quarter. It also lead us to having to refactor the project mid-way through in order to provide something that was up to both our standards, as well as the sponsors.

The spiral methodology that we chose to use would have worked better if we had been able to correctly estimate what we would be able to accomplish in an iteration. Our inexperience with the technologies made us either under or overestimate the amount of work that we could get done in an iteration, which lead to having too much or too little expected from us in a single iteration. This would, most likely, have been a problem for us no matter what we chose for a methodology.

Overall, while we would have been happier if we could have gotten more functionality complete, the project was a success, and our sponsor was happy with how the project turned out. Some aspects of our design and implementation have also made our customer extremely happy, so that makes the project seem worthwhile.