**Rationale DB Design**

ClubUML Application

2/16/2014

Addresses use case RM1

**Overview**

The Rationale entity is a new entity added during the Spring2014 semester. It replaces the previously used Comment entity with a more detailed explanation for diagram promotion. With this addition, the use can enter details about how a particular UML diagram may better address project requirements or better prevent bugs. This information should be persisted so that a user can review aggregate rationales and make more informed decisions while choosing between diagrams.

**Database Technical Details**

ClubUML’s current database is implemented in MySQL. Because the Rationale entity is a simple addition, we will continue to use MySQL with JDBC access mechanisms and a very similar database schema. All of the existing tables retain their attributes and constraints. The exception to this is the Comment entity, which has been removed. Because this system is not yet fielded, we can remove the comment entity without transferring existing comments to the new database schema.

**Data Design**

The new Rationale data expands the initial comment data to include more specific reasoning-based attributes. The table below briefly describes the Rationale entity attributes and identifies the primary and foreign keys.

|  |  |  |  |
| --- | --- | --- | --- |
| Rationale Table |  |  |  |
| **Description** | This table describes the reason a user promoted some diagram over another. | | |
| **Attribute** | **Description** | **Type** | Examples of values |
| rationaleId | Id of this rationale instance | INT(11) | -2147483648 to 2147483647 |
| compareId | Id of the rationalized comparison | INT(11) | -2147483648 to 2147483647 |
| userId | Id of user creating the rationale | INT(11) | -2147483648 to 2147483647 |
| promotedDiagramId | Id of promoted (favored) diagram | INT(11) | -2147483648 to 2147483647 |
| alternativeDiagramId | Id of alternative (un-favored) diagram | INT(11) | -2147483648 to 2147483647 |
| rationaleTime | Time when rationale was created | TIMESTAMP | '2014-02-16 00:00:01' UTC |
| summary | Brief overview of reasoning behind promotion | VARCHAR(255) | ‘Diagram A is better diagram because…’ |
| issue | Name of issue affecting rationale | VARCHAR(75) | ‘Bug #5’ |
| issueRelationship | Description of how issue affects the rationale | VARCHAR(255) | ‘Diagram A better prevents Bug #5 because…’ |
| criteria | Name of criteria affecting rationale | VARCHAR(75) | ‘Requirement #6’ |
| criteriaRelationship | Description of how the criteria affects the rationale | VARCHAR(255) | ‘Diagram A better addresses Req. #6 because…’ |
| Primary Key | rationaleId | | |
| Foreign Keys | compareId, userId, promotedDiagramId, alternativeDiagramId | | |

**Relationships**

The Rationale entity has four relationships with three existing database entities. These relationships are specified via the foreign keys that identify the related entity. All of these relationships are many-to-one relationships because the rationale only refers to one each of compare, user, promoted diagram, and alternative diagram. But each of these may have many associated rationales. Additionally, all of these relationships are non-identifying.

Foreign Keys:

* **compareId**- The compareId key identifies a compare entity. The compare entity holds a comparison that was made between two existing diagrams, so there is a separate compareId for each pair of diagrams. The rationale entity describes how a user rationalizes the value of a certain diagram in the context of some comparison, so the compareId identifies that comparison context. This relationship is a many-to-one relationship because each rationale can only refer to one comparison, but a comparison may have several rationales.
* **userId**- The userId identifies a user entity which is used to track user activity when necessary. This id can be used to identify who created the rationale. This relationship is again many-to-one because a user may contribute several rationales across the project, but a single rationale must be created by a single user.
* **promotedDiagramId**- The promotedDiagramId refers to a diagram entity. A diagram entity holds all of the relevant metadata about a singular diagram that has been uploaded to the application. The promotedDiagramId identifies the diagram that the user favors in this rationale.
* **alternativeDiagramId**- The alternativeDiagramId also refers to a diagram entity. This identifies the diagram, which is not favored by this rationale.

These relationships can be seen more clearly in the EER diagram below, which visually describes how the new rationale entity fits into the current database design. Please note that this does not include all entities, only those that are immediately related to the rationale entity.



**SQL Scripts**

SQL scripts can be used to configure the database to accept the new type of data.

If a developer needs to remove the comment entity, they can use the following short SQL script:

Use clubuml;

DROP TABLE comment;

From there, the developer can create the rationale entity using the following table creation script:

-- Table rationale

Use clubuml;

CREATE TABLE rationale

(

rationaleId int(11) NOT NULL PRIMARY KEY AUTO\_INCREMENT,

compareId int(11) NOT NULL,

userId int(11) NOT NULL,

rationaleTime timestamp NOT NULL,

promotedDiagramId int(11) NOT NULL,

alternativeDiagramId int(11) NOT NULL,

summary varchar(255) NOT NULL,

issue varchar(75) NOT NULL,

issueRelationship varchar(255) NOT NULL,

criteria varchar(75) NOT NULL,

criteriaRelationship varchar(255) NOT NULL

);

ALTER TABLE rationale ADD CONSTRAINT rationaleHaveCompareId FOREIGN KEY (compareId) REFERENCES compare (compareId) ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE rationale ADD CONSTRAINT rationaleHaveUserId FOREIGN KEY (userId) REFERENCES user (userId) ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE rationale ADD CONSTRAINT rationaleHavePromotedDiagramId FOREIGN KEY (promotedDiagramId) REFERENCES diagram (diagramId) ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE rationale ADD CONSTRAINT rationaleHaveAlternativeDiagramId FOREIGN KEY (alternativeDiagramId) REFERENCES diagram (diagramId) ON DELETE NO ACTION ON UPDATE NO ACTION;

**Data Access**

All of the rationale attributes can be created or read by the application. The IDs and timestamp are created by the application itself, and the five text inputs are created directly by the user via the rationale dialog box. The user can view this information via the rationale view located beneath the diagrams on the compare page. We control access to the database via the Rationale data access object (DAO). The DAO is implemented in Java to provide methods to control rationale data access and provide convenient methods to the rest of the application. This currently includes methods to add rationale and get rationale data from the database, but the DAO can easily be extended in the future to include update and delete operations. The create and read operations are done using JDBC, so we directly use SQL prepared statements in the java DAO code as follows:

Create SQL Query:

INSERT INTO rationale(compareId,userId,summary,issue,issueRelationship,criteria,criteriaRelationship,rationaleTime,promotedDiagramId,alternativeDiagramId) VALUES(?,?,?,?,?,?,?,NOW(),?,?);"

Read SQL Query:

"SELECT \* FROM rationale where compareId = ? ORDER BY `rationaleTime` DESC;"