DATABASE REFACTORING IN THE PRESENCE OF MULTIMEDIA FUNCTIONAL DEPENDENCIES

JOSHUA GEIGER

CS 2310 MILESTONE #1

INTRODUCTION

Refactoring databases is defined by Ambler and Sadalage [1] as the process of applying simple changes improve the database schema design without altering its behavioral or informational semantics. These two men have been instrumental in determining the best practices for database refactoring including providing a catalog of over 60 specific refactoring techniques across 5 major categories of refactorings. Applied refactorings cannot change the interpretation of the data in the DB nor can it add or remove functionality. Since the schema is altered, we must additionally update the how our programs access the database, i.e. rewrite their queries to utilize the new schema. Refactoring has many goals, one of which is normalization.

Normalization is a means of ensuring the database is structured such that queries are efficiently processed and no anomalies exist that could endanger data integrity. Functional dependencies between attributes form the basis of the whole normalization process. To normalize the database, we systematically apply refactorings until our schema reaches one of the six normal forms, most commonly the third normal form which will eliminate the data integrity anomalies we wish to avoid [2]. With the addition of multimedia attributes to databases, normalization becomes more difficult.

Chang et al. proposed a new type of functional dependency called the multimedia functional dependency, which in turn introduces new multimedia normal forms designed to specifically eliminate anomalies arising from the additional multimedia attributes [3]. These normal forms will require different refactoring techniques to normalize the database than traditional databases.

SOME BASIC REFACTORINGS

Below are described three important refactoring techniques used in the normalization of databases.

MOVE COLUMN

The basic idea of *Move Column* is to migrate a table column, with all of its data, to another existing table. Often a column in a table will break a normalization rule, so by moving it to another table, we can increase the source table's degree of normalization and reduce redundancies in the database.

Ambler and Sadalage point out the following necessary actions for updating the schema:

1. Identify deletion rule(s). Determine what action should occur if a row in one table is deleted. For example, do we null or zero the values of the corresponding row in the other table or do we delete it as well?
2. Identify insertion rule(s). Similarly to identifying the deletion rules, we must determine what happens when a row is inserted into one of our tables and how it impacts the other.
3. Introduce the new column. Use the SQL command ADD COLUMN to create the column in the table.
4. Introduce triggers. We must install triggers to copy data from one table to another during a period called *transition* where both schemas must be valid since not all accessing programs may be updated yet.

After altering the schema, the data in the old column must be copied to the new. Now the only thing left is to update the how our program accesses the DB. Once again Ambler and Sadalage enumerate the necessary steps:

1. Rework the joins to use the moved column.
2. Add the new table to joins.
3. Remove the original table form joins.

After completing the modification of all of the program accesses, you will be able to remove the triggers we added to ensure the old schema was still valid.

SPLIT TABLE

This refactoring is used to divide a table vertically into one or more tables. Similar to *Move Column* the goal here is to remove normal form breaking columns so that our original table is normalized. To actually perform the schema modification, we use the CREATE TABLE command and repeatedly perform the *Move Column* being sure to add triggers to synchronize our columns. While doing this it is important not to inadvertently set cycles between tables.

Updating the program accesses involves updating any table metadata our program uses or embedded SQL queries. Finally, to allow our users to take advantage of the update, we should update the user interface.

MERGE TABLES

The *Merge Tables* technique is used to combine multiple tables into a single table. The schema is updated by using ADD COLUMN to introduce the columns of one table into table we wish to merge. Since the merged table may already contain some of the columns, inconsistencies and clutter can arise that will need to be sorted out. Triggers must be set again to ensure synchronization among tables during the transition phase. We migrate the data by copying all the data from the original table into the merged one. After the transition period, we can drop the old table and the associated triggers.

PROJECT

Ideally it would have been nice to update all 60+ refactoring techniques Ambler and Sadalage provide in their refactoring catalog to handle multimedia dependency aware. Even though that is far outside the realm of possibility, this project still intends to accomplish these goals:

I. Develop, in detail, three multimedia-aware refactoring techniques (*Move Column, Split Table,* and *Merge Table*) for use with a database containing MFDs. Focus on not only how to refactor, but why, i.e. what triggers the need to refactor and what do we hope to gain my performing this refactoring. Ideally an algorithm to automate the refactoring operation can be developed, but a detailed guide for performing these refactorings like those provided by Ambler and Sadalage may be all that is possible.

II. Provide examples of the multimedia aware refactoring techniques.
    a. Example tables containing multimedia elements using:
        i. Original schema
        ii. Transition schema
        iii. Resulting schema
    b. Example schema update SQL for the refactorings
    c. Example data-migration SQL
    d. Synchronization triggers used in the example the during transition phase

III. Utilize these techniques to normalize an example multimedia database to some MNF.

REFERENCES

[1] Ambler, S. W. and Sadalage, P. J. 2006 *Refactoring Databases: Evolutionary Database Design*. Addison-Wesley Professional.

[2] Elmasri, R. and Navathe, S. B. 2006 *Fundamentals of Database Systems (5th Edition)*. Addison-Wesley Longman Publishing Co., Inc.

[3] Chang, S., Deufemia, V., Polese, G., and Vacca, M. 2007. A Normalization Framework for Multimedia Databases. *IEEE Trans. on Knowl. and Data Eng.* 19, 12 (Dec. 2007), 1666-1679. DOI= http://dx.doi.org/10.1109/TKDE.2007.190651