International Journal of Innovative
Computing, Information and Control
Volume **10**, Number **6**, December **2014**

 ISSN 1349-4198
pp. 2209–2220

A WORKFLOW FOR DATABASE REFACTORING

Marcia Beatriz Pereira Domingues, Jorge Rady de Almeida Junior
Wilian França Costa and Antonio Mauro Saraiva

Polytechnic School
University of Sao Paulo
Av. Prof Luciano Gualberto, Trav.3, n.158, São Paulo, SP, 05508-010, Brazil
{ marciabeatrizcp; wilianfc }@gmail.com; jorge.almeida@poli.usp.br; saraiva@usp.br

Received December 2013; revised April 2014

Abstract. *The development and maintenance of a database are significant challenges due to frequent changes and requirements from users. To follow these changes, the database schema suffers structural modifications that, many times, negatively affect its performance and the results of queries, such as unnecessary relationships, primary and foreign keys created and strongly attached to the domain, with obsolete attributes or inadequate types of attributes. The literature on Agile Methods for Software Development suggests the use of refactoring for the evolution of database schema when there are requirement changes. A refactoring is a simple change that improves the design, but it does not alter the semantics of the data model or add new functionalities. This workflow has significant differences in relation to that proposed by Amblers, and its use brings significant advantages in the maintenance of complex databases with large amounts of data. As a case study, a relational database, used by an information system for precision agriculture, was used. This system is web based and needs to archive data, retrieve information, and respond to large geospatial queries.*

Keywords: Evolutionary databases, Database refactoring, Database schema, Query performance, BPMN

1. **Introduction.** The difficulty of giving support to the schema's evolution has been recognized as one of the main obstacles to updating information systems [10]. Studies show that these difficulties have increased with the use of Web systems, where the changes happen more frequently, and there is very little tolerance to errors and unavailable services.

Refactoring a database requires more work than refactoring an application's source-code; however, according to Fowler et al. [10], it can follow the same premises. It is necessary to be concerned with all steps of the applications that use the database, to perform changes without disregarding the user's requirements, to preserve the existing data, and to not make any changes to the semantics of the actual schema and maintain the database integrity.

The techniques for code refactoring were proposed by Flower et al. [10] in the Agile Methods for Software Development literature and were adapted for database by Ambler and Sadalage [1,2]. Ambler and Sadalage defined refactoring as a small change to the schema that improves its design but does not add functionalities or alter its informational or behavioral semantics [2].

However, not all changes fit into that refactoring definition. The necessity of implementing large changes in databases is very common. It implies choosing a set of refactoring that represents the desired database final schema, providing the opportunity to achieve the necessary planning to perform a large change. This set of refactoring requires previous knowledge of the actual database, as well as its limitations and objectives. Ambler has

shown which refactorings are possible as well as their tasks; however, he considers that the planning of the execution of these refactorings should be performed in a sequential manner. Although, if said refactoring were executed in a workflow, we can see that many tasks could be performed in an independent manner, in parallel and combined with similar refactoring tasks.

The main objective of this work is to propose a new workflow that applies to a set of refactoring tasks in a database schema. The case study exposed here reinforces how difficult it is to abstract the domain of the database model. This dependency of the domain damages the model design and leads to problems that can negatively affect the query performance. The database refactoring workflow applied in the case study aims at the query performance gain and design improvement, but the objective of the case study is to validate each activity of the workflow to show that it is correct and secure to apply in any database schema.

The case study uses a relational database employed by an information system for precision agriculture. This system is web-based and needs to perform large data transfer to graphics with geo-referenced information [13]. This database has many modeling problems in its schema, which suggests a refactoring process to improve query performance. Here, we present some phases of the refactoring process, as well as its positive impact on time consumption during query execution.

Materials and methods. A spatial database of an existing Precision Agriculture Information Portal System was used to test the database's refactoring workflow. A PostGIS extension was used to provide spatial objects for the PostgreSQL database in order to store and query the information about location and mapping. The refactorings called "Introduce Surrogate Key", "Merge Column", and "Introduce Index" were chosen from Ambler's Database Refactoring Catalog. In the proposed experiment, the costs and time consumed for the queries were considered to compare the performance between the original model and the refactored model. The tests performed included new codes in the database function language (plpgsqq) for data insertion, selection, and cascade delete operations of geographic data through geospatial queries.

Paper organization. Section 2 presents the research related to database refactoring, database evolution and limits of the existing solutions. Section 3 presents the proposed workflow. Section 4 is dedicated to show how an existing PA Information Portal System was refactored to improve query performance and time results. Section 5 shows the results of the experiment, and Section 6 presents our conclusions and final remarks.

2. Related Work.

Ambler wrote a catalog of refactoring with 49 types of refactoring divided into 4 types: architectural, structural, data quality and referential integrity [1,2]. Ambler discusses for each refactoring the motivations, implications and cautions for its operation, showing the steps for the possible changes on the database. These steps were adapted in this paper in order to develop the Refactoring Workflow. Some refactoring from Ambler's catalog require data replication. Ambler proposed a synchronous solution, whereas Domingues et al. [8] proposed an asynchronous one.

On Ambler's and Domingues's techniques, there are limitations to automatize a group of refactoring, once their process had only been tested on singular refactoring processes, thus, their work did not foresee grouped tasks.

The tool developed by D'Souza and Bhatia [9] adopts a template design pattern to make the database independent of the method. In this model, D'Sousa developed a tool called "The Metadata Manipulation Tool", which uses metadata to acquire the actual data schema, including its primary and foreign keys. D'Sousa emphasizes the limitation of working with generic databases and databases that answer to multiple applications.

Boehm et al. [3] also used metadata to develop a tool called Squash (SQL Query Analyzer and Schema EnHancer), which visualizes the actual schema and analyzes modeling problems that can be solved with the tool, followed by the development of queries in SQL to be applied in the correction of the schema. The entire schema's representation was based in XML (eXtensible Markup Language). The tool measures queries execution times after refactoring the schema and adding indexes. The work does not consider other types of refactoring, but the developed tool is a good option for data schema analysis.

Chang et al. [5] proposed a database refactoring framework that analyzes the changes to the database schema and the impact of these changes on the queries. The framework defines a logical model of changes and tries to find and solve the model's inconsistencies and modeling problems. This work is a recommended step for the process of automation of database refactoring. Chang formalizes the database refactoring problem using Predicate Logic and emphasizes the necessity of amplifying the refactoring process to make it capable of making decisions and to be more independent and uses the proposed process in a more visual language.

Curino et al. [7] presents a tool called PRISMA, which helps DBAs (database administrators) to predict and evaluate the effects on the applications due to changes to the database. It also executes the changes selected by the DBAs and saves all the changes registries. Curino emphasizes the positive effects of refactoring on the database for queries performance and the necessity of database evolutions using workflows.

Among the free and commercial tools, Liquibase [12], written in JAVA, works in any database to track, manage and apply the changes on the database. All changes are stored in a way that is easy to read and to track in a version controller. This tool is constantly updated by the free software community to answer to new versions of databases; however, there are still limitations for this tool in the case of geographic databases.

Table 1 compares the presented related works. None of the works presents a workflow to execute a large change in a database. Two works from Ambler and Curino present

TABLE 1. Comparison of researches for database refactoring

	Ambler	Domingues	D'Souza	Boehm	Chang	Curino	Liquibase
Automation	No	No	No	No	Limited	No	No
Generic databases	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Concurrency control	Yes	Yes	No	No	No	No	No
Number of refactoring per process	Limited	Limited	Limited	Limited	Limited	No	No
Data Replication	Yes	Yes	No	No	No	No	No
Tool	No	Database Evolution Manager	No	Squash	No	Prisma	Liquibase
Language and Structure	No	No	JAVA Metadata	XML	No	XML	JAVA XML
Improving Database design	Yes	Yes	No	Yes	Yes	Yes	No
Improving query's performance	No	No	Yes	Yes	Yes	No	No
Changes' history	No	No	No	No	No	Yes	Yes
Workflow	Yes	No	No	No	No	Yes	No
Refactoring Suggestion	Yes	No	No	No	No	No	No

a workflow, but both are limited and the workflow of Ambler presents many problems, which we have solved. The advantages of our workflow against Amblers will be detailed in the next section.

3. **Proposed Workflow.** This workflow starts from the original database schema, in which the user needs to improve the query performance or the database's design model. The workflow receives information defined in this work as modeling problems as well as information to execute changes according to the user's requirements. Figure 1 shows the workflow's context transforming the initial schema in the final one, starting from the users' requirements and collected problems.

Ambler proposed the refactoring schema presented in Figure 2. To execute a set of refactoring and to solve many problems from the previous workflow, we developed the workflow presented in Figure 3. Below we describe each activity of our workflow.

1. **Save original schema**: Before all refactoring, it is necessary to back up the schema that will be changed. This is necessary in any environment (development, test or production). Even in the development environment, it is important to make a backup in order to recover the changed schema.
2. **Suggest a set of refactoring**: The start of the process is the grouping of the user's requirements and a list of the problems identified in the production environment. Both have to be translated to a refactoring list; this task has to be done by the database administrator that knows the structure and the business's domain. The mapping of a requirement or the solution of a problem in a refactoring has great influence from the database administrator, but the process makes it possible for an

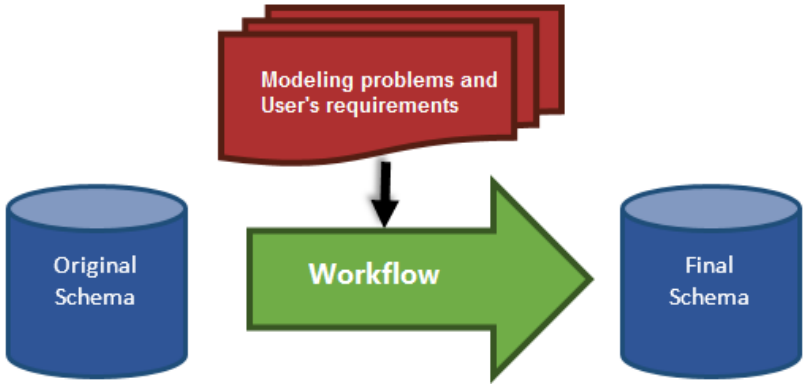

FIGURE 1. Workflow transforming the original schema into the final schema

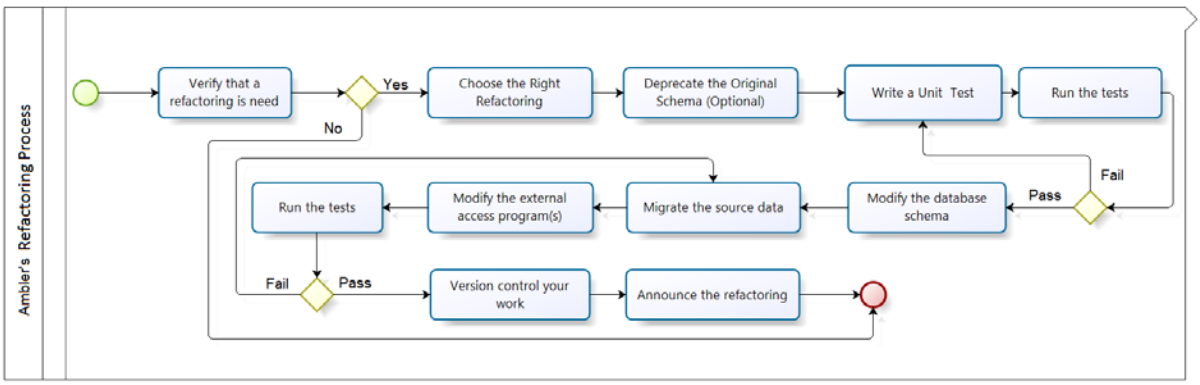

FIGURE 2. Ambler's workflow

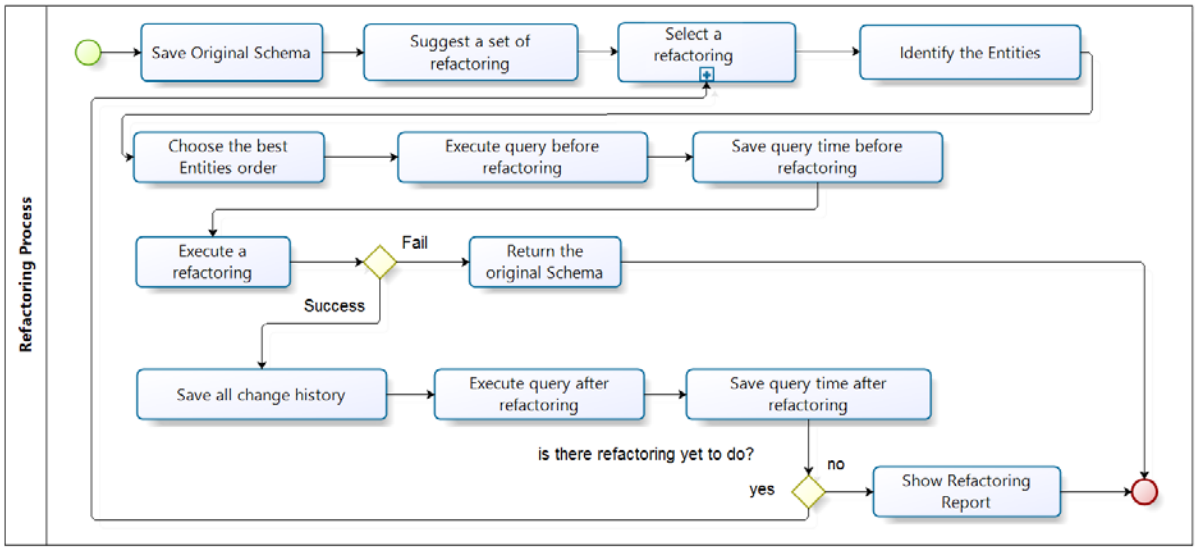

FIGURE 3. The new workflow

administrator without much experience to make a refactoring task and verify if the result was the one he expected.

3. **Select a refactoring**: The administrator has to start by selecting a refactoring task from a list of pre-selected refactoring tasks. The administrator only has to make sure that the chosen refactoring task has all the necessary pre-requirements. It is not necessary, at first, to choose the best execution order because the best order will depend on the environment, on the database, and on the type of requirement or problem that is being solved. Because the refactorings are executed in the environment (development, test, and production), the administrator will gain experience to re-order the refactoring to save time.
4. **Identify the entities**: Many refactorings require changes in different entities. In those cases, the administrator, knowing the application, must list all entities involved in the refactoring, and to execute the next task, also document the dependencies between them.
5. **Choose the best entities order**: At this moment, it is necessary to order the entities that will be changed during the refactoring. The best order is the one that allows all entities to be altered without losing their integrity, their data, and their references.
6. **Execute query before refactoring**: For each entity involved in the refactoring, it is necessary to write queries that bring the data from their tables and also from parent and child tables (the ones that refer and the ones that are referenced). One possibility to automatize this task is to collect from the database manager log, for a representative period, all queries made to the database. Identify the most common and the most complex one that should be used in this task.
7. **Save query time before refactoring**: Even if the refactoring does not have the objective of improving the database performance, it is important to collect and save the queries execution time before any refactoring is applied. The objective is to measure the impact of the refactoring on the databases performance. If the objective is not to improve the performance, it is still important to not make the databases performance worse.

8. **Execute a refactoring**: This task will change the original schema of the database toward the expected result. Each refactoring will demand a very specific group of actions. Thus, the administrator must read very carefully all the instructions available on Ambler's [2] refactoring catalog or Domingues' [8] reorganized catalog, in Portuguese. In BPMN, this activity is a subprocess of Figure 3.
9. **Return the original schema**: This activity will only be executed if there is a problem in the execution of the refactoring. To guarantee the safety and the integrity of the data, the entire process will be canceled, regardless of how many successful refactorings had been made.
10. **Save all change history**: All scripts executed in the database should be saved to make their execution possible in other environments. In addition, it is important to save all scripts to make an audit possible in the case of a future problem. The complexity of this task can vary accordingly with its necessity. It can simply be an act of protection from the database administrator. Thus, this task means to save the script in a code version manager (e.g., SVN or Git). On the opposite extreme, it can be necessary that before the previous task is executed (executing a refactoring) to follow a change process in the company.
11. **Execute query after refactoring**: After the refactoring, the entire group of queries to the database should be executed. The successful execution of this task means that the refactoring was performed according to plans. It is possible that a specific query fails due to the changes, but its execution is important. The error in this query means that the refactoring has been made, causing the error. A group of errors, at the end of the process, can be used to exemplify what the developers cannot do anymore in their applications.
12. **Save query time after refactoring**: For those queries that were successful, it is necessary to collect the execution time or any other measure system (e.g., execution plan) to make it possible to verify the impact on the performance of the database.
13. **Show refactoring report**: This activity is executed when the whole set of refactoring defined on the workflow has been executed. The objective is to compile all existing information in the workflow to write a report. The main information that should be part of this report are:
    - List of all refactorings made and their order;
    - The scripts that changed the database;
    - Scripts that have all the queries executed before and after each refactoring;
    - Time consumed before and after those queries;
    - A list of observations, comments and alerts of the problems that were found.

    The objectives of this report are to make an analysis of the impact of the workflow, to document everything that was done and to communicate the new database schema to all involved in the process.

Advantages of the proposed workflow. The proposed workflow (Figure 3), offers the following advantages.

1. It works with a large change, once it is concerned with the definition of a set of refactorings and the execution of all of them. The previous model does not concern itself with this aspect, even though it is documented that many refactorings depend on others to make their execution possible.
2. The first task of the new process is to back-up the schema that will be changed. In the previous process, this concern is defined in another moment, considering if the environment is of production or not. The new process makes this task necessary in all environments.

3. Deals exclusively with the tasks of the database administrator, not with the application developer, such as "modifying the external access program(s)".
4. Instead of a generic task called "write a unitary test", the new workflow considers it is necessary to write a group of queries for all entities involved in the refactoring. This group of queries can be obtained from the logs of the database manager.
5. The task collects the time consumed by a group of queries before and after the refactoring. This concern with time consumed is very important to all databases.
6. Unifies the tasks "modify the database" and "migrate data" into a single task named "execute the refactoring". The refactoring documentation will inform if it is important to worry about the migration of the data and the available options. This process is important because only 20% of the refactorings demand data migration, according to Domingues [8].
7. The generic task "announce a refactoring" is replaced by a well-defined task, which is "divulge a refactoring report". This report will be the basis for the decision if the process was successfully executed and if it is viable to continue the refactoring process in another environment.
8. The new workflow was written in BPMN, which makes it possible to validate, evaluate, improve, and automatize. These are characteristics that offer safety to the process, because it is important to assure that everything executed in one environment can be repeated in another.
9. Concerns itself with the order of the involved entities. A practical problem of any database change that involves more than one entity is which entity should be changed first. This relevant question should be solved before the refactoring is executed.
10. Lastly, because the back-up was made at the beginning of the process and a large set of refactoring could be executed, if any refactoring from this group fails, or any problem that could affect the process occurs, there is the task of recovering the previous schema and finalizing the process.

4. **Case Study.** A spatial database of an existing Precision Agriculture Information Portal System (PA System) was used to test the database refactoring methods. For this domain, Information Systems can improve farm management, helping to make the best decisions based on all available information; maintaining, controlling, and optimizing resources and returns; and preserving the natural environment [11]. Management and decision support systems should be designed to meet the specific needs of the farmers [4]. Crop analysis is usually performed considering the specific characteristics of each plant population, soil, crop and the climate conditions [14].

A PostgreSQL database was used with the PostGIS extension to provide spatial data type and functions in order to store and retrieve information about location and mapping. This database and its programed functions on PostGIS were used as part of the solution for the development of processing web-services, publishing and the dynamic visualization of agricultural data through a map server [13].

The system needs to archive, retrieve, and process data for future analyses without neglecting aspects of the entity-relationship model [6]. The original PA Information Portal System database model is presented in Figure 4, where there are many relationships that describe the analyzed crop cycle stages [13].

The original database model was built with composite natural keys. There is a strong coupling between the database model and the Precision Agriculture business domain and some primary key columns, such as farm id, plot id, and prod id, that are present in almost all relations. Figure 5 shows the tables farm, plot, productivity and productivity

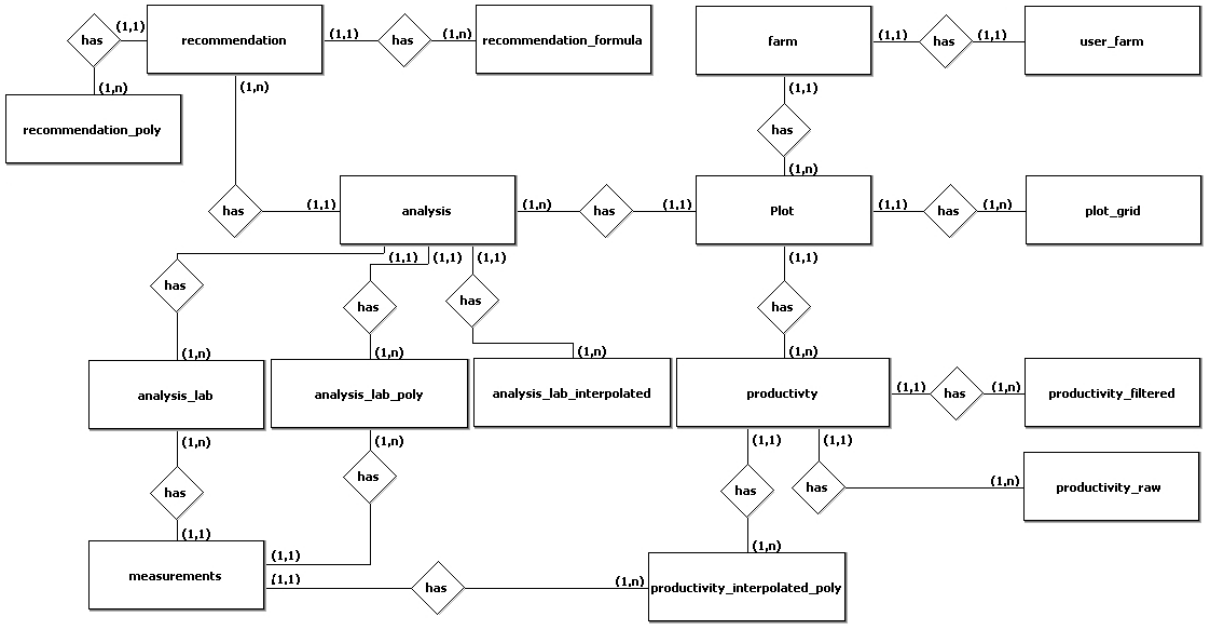

FIGURE 4. Original database model

raw data from the original model before refactoring. These tables represent the most important database relations that can be present as:

- Plot: A plot contour represents a parcel of the farm that is used as a crop field. This entity must have a polygon associated due to the fact that this information is used for geoprocessing methods (e.g., machine productivity filtering and interpolation methods).
- Farm: Represent the set of plots used as agricultural crop fields. It can be represented by multipolygonal geometry (not mandatory).

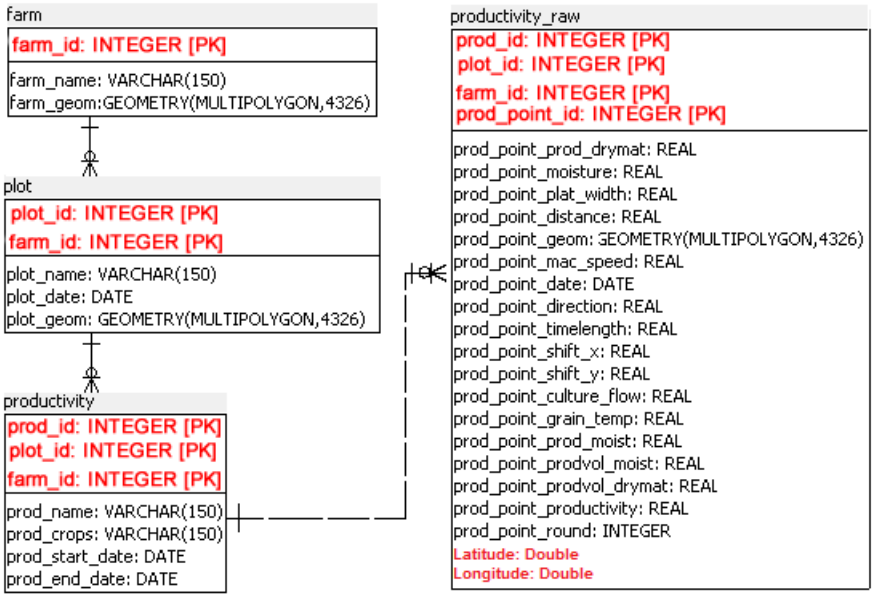

FIGURE 5. Tables (farm, plot, productivity and productivity_raw) before refactoring

TABLE 2. Refactorings applied in the case study

	Description	Type	Impact
Introduce surrogate key	Multiple attributes to the primary key	Structural	Design and performance
Add Foreign Key Constraint	Guarantee that the attributed values are preexistent	Referential Integrity	Design
Merge Columns	Separate columns that together represent one information only	Type	Design and performance
Introduce Index	Enhance queries performances	Architectural	Performance

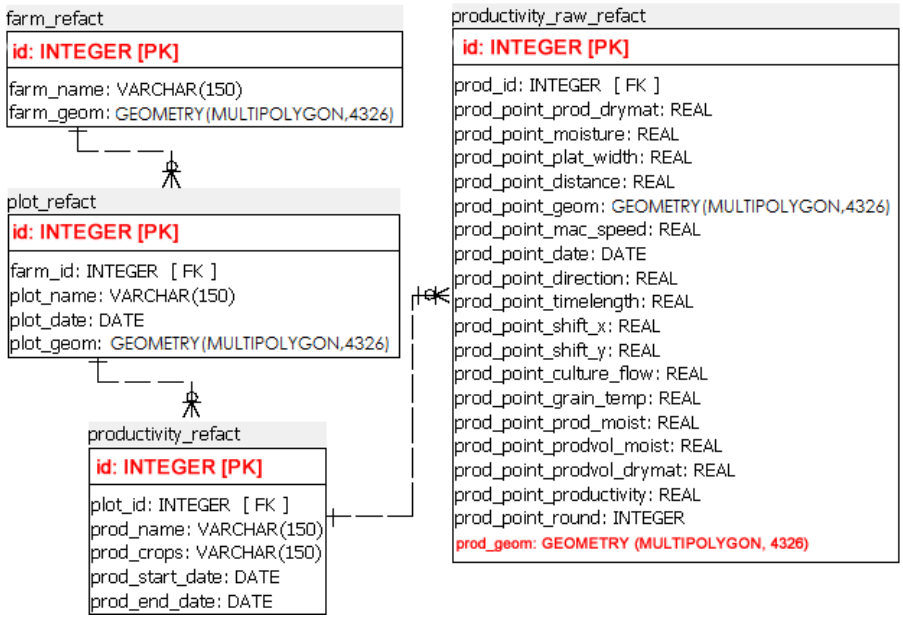

FIGURE 6. Tables (farm, plot, productivity and productivity_raw) after refactoring

- Productivity: The productivity represents the harvest for a crop field. This entity represents only the facts (e.g., harvesting start date). The real data points are represented in the entities productivity raw and productivity filtered.
- Productivity_raw: The productivity raw entity has the data collected by the harvesting machine as obtained (i.e., without any type of data processing).

Based on the workflow presented in Figure 3 and all the problems of the original schema, we have applied four refactorings described in Table 2.

After all refactorings of Table 2 have been completed, we have the final schema in Figure 6.

5. **Results.** The refactoring called "Introduce Surrogate Key" was chosen from Ambler's Database Refactoring Catalog. In the proposed experiment, the time consumed for the queries was used to compare the performance between the original model and the refactored model. The tests performed included new codes in the database function language (plpgsqq) for data insertion, selection, and spreading of deleted operations of geographic data through geospatial queries.

To allow a direct comparison between original and refactored model, the consumed time (ms) was measured. After the refactoring, the time consumed was, 15% shorter to insert, 25% shorter to select, and 4% shorter to delete data for 189.730 rows, from both

original and refactored databases. In Figure 7 we highlight that the refactored database performance was better than in the original, considering the time results.

In the present case study, we also performed the "Merge Column" refactoring in order to merge the latitude and longitude columns of the product_raw table. To perform this refactoring, a new geometry column, **prod_geom**, was introduced with the previous data from the latitude and longitude columns.

The select operation after this merge was 20% better in terms of time consumed to select the data operation than in the original model. The new merged column can be a candidate to be indexed, but when "Introduce Index" refactoring was applied, the time consumed to select the data increased by 3%. The results for "Merge Column" and "Introduce Index" refactorings are shown in Figure 8.

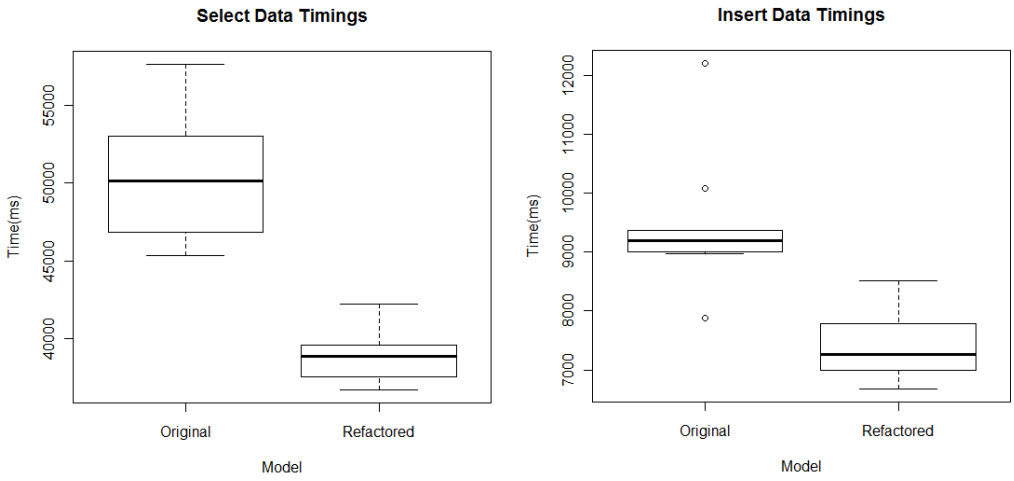

FIGURE 7. Time results for insert and select data

FIGURE 8. Time results for data selection after "Merge Column" and "Introduce Index" refactorings

6. **Conclusions.** Database refactoring is a technique for restructuring an existing database schema in order to respond to new system requirements, improve schema design or improve query performance.

The present work aims to contribute to the existing research by proposing a new workflow for refactoring databases, which solves many of the problems of the previous workflow and adds the possibility to execute a set of refactorings in the database.

We used the new workflow in the case study to execute a set of four refactorings. Comparing the original and refactored database performance, higher scores were obtained by the refactoring tasks. For the "Introduce Surrogate Key" refactoring, the measured time consumed (ms) to insert, delete, and select data was 15%, 4%, and 25% shorter, respectively. The "Add Foreign Key Constraint" refactoring improved the data integrity of the schema, and the "Merge Column" improved the selection performance by 20%. Only the "Introduce Index" did not result in an improvement. That is why we discarded this refactoring process for this case study.

The proposed refactoring database workflow has many advantages, and it was successfully executed in the case study. Because in this workflow, a set of four refactorings could be executed, all the performance data were collected before and after each refactoring. Three of the refactorings were applied in all the application environments.

Work contributions. Based on what has been presented, we can list some of this contributions of this work for the study of database evolution using refactoring:

- This work presents a defined process in BPMN to perform refactoring in database.
- The process has important activities that were not well-described in Ambler's refactoring process, such as back-up of the original system, time consumed in query collection before and after the refactoring, and the orders the entities involved in the refactoring.
- The workflow considers a set of refactorings and instead of isolated ones. This set is named in the work as a large change on the database.
- The validation of the workflow was performed in a database with a large volume of data that had all the requirements to perform a large change.
- The performed refactoring in the case study presents performance gains to the system.

REFERENCES

[1] S. W. Ambler, *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*, John Wiley and Sons, 2002.

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

[3] A. Boehm, D. Seipel, A. Sickmann and M. Wetzka, Squash: A tool for analyzing, tuning and refactoring relational database applications, in *Applications of Declarative Programming and Knowledge Management*, D. Seipel, M. Hanus and A. Wolf (eds.), Springer Berlin Heidelberg, 2009.

[4] R. Bongiovanni and J. Lowenberg-Deboer, Precision agriculture and sustainability, *Precision Agriculture*, vol.5, no.4, pp.359-387, 2004.

[5] S.-K. Chang, V. Deufemia, G. Polese and M. Vacca, A logic framework to support database refactoring, in *Database and Expert Systems Applications*, R. Wagner, N. Revell and G. Pernul (eds.), Springer Berlin Heidelberg, 2007.

[6] P. P. Chen, The entity-relationship model – Toward a unified view of data, *ACM Trans. Database Syst.*, vol.1, no.1, pp.9-36, 1976.

[7] C. Curino, H. J. Moon and C. Zaniolo, Automating database schema evolution in information system upgrades, *Proc. of the 2nd International Workshop on Hot Topics in Software Upgrades*, New York, NY, USA, pp.1-5, 2009.

[8] H. H. Domingues, F. Kon and J. E. Ferreira, Asynchronous replication for evolutionary database development: A design for the experimental assessment of a novel approach, *Proc. of Confederated*

International Conference on the Move to Meaningful Internet Systems – Volume Part II, Berlin, Heidelberg, pp.818-825, 2011.

[9] A. D'Sousa and S. Bhatia, Refactoring of a database, *International Journal of Computer Science and Information Security*, vol.6, no.2, pp.307-315, 2009.

[10] M. Fowler, K. Beck, J. Brant, W. Opdyke and D. Roberts, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional, 1999.

[11] R. Khosla, Precision agriculture: Challenges and opportunities in a at world, *The 19th World Congress of Soil Science*, 2010.

[12] Liquibase, *MS Windows NT Kernel Description*, http://www.liquibase.org, 2013.

[13] E. Murakami, A. M. Saraiva, L. C. M. Ribeiro Junior, C. E. Cugnasca, A. R. Hirakawa and P. L. P. Correa, An infrastructure for the development of distributed service-oriented information systems for precision agriculture, *Comput. Electron. Agric.*, vol.58, no.1, pp.37-48, 2007.

[14] F. S. Santana, E. Murakami, A. M. Saraiva and P. L. P. Correa, A comparative study between precision agriculture and biodiversity modelling information systems, *Proc. of the 6th Biennal Conference of the European Federation of IT in Agriculture*, 2007.