Journal of Database Management
Volume 27 • Issue 1 • January-March 2016

Dealing with Dangerous Data: Part-Whole Validation for Low Incident, High Risk Data

Cecil Eng Huang Chua, Information Systems and Operations Management Department, University of Auckland, Auckland, New Zealand

Veda C. Storey, Department of Computer Information Systems, J. Msck Robinson College of Business, Georgia State University, Atlanta, GA, USA

ABSTRACT

In certain situations, syntactically valid, but incorrect, data entered into a database can result in near-immediate, catastrophic financial losses for an organization. Examples include: omitting zeros in prices of goods on e-commerce sites; and financial fraud where data is directly entered into databases, bypassing application-level financial checks. Such "dangerous data" can, and should, be detected, because it deviates substantially from the statistical properties of existing data. Detection of this kind of problem requires comparing individual data items to a large amount of existing data in the database at run-time. Furthermore, the identification of errors is probabilistic, rather than deterministic, in nature. This research proposes part-whole validation as an approach to addressing the dangerous data situation. Part-whole validation addresses fundamental issues in database management, for example, integrity maintenance. Illustrative and representative examples are first defined, and analyzed. Then, an architecture for part-whole validation is presented and implemented in a prototype to illustrate the feasibility of the research.

KEYWORDS

Audit, Boyce-Codd Normal Form, Business Rules, Dangerous Data, Data Management, Data Quality, Database Design, Part-Whole Validation, Relational Databases

INTRODUCTION

Data availability and reliability has always been of upmost importance for managerial decision making. It is increasingly so as the reliance on real-time, large volumes of data increases (Batini, Rula, Scannapieco, & Viscusi, 2015). Data management is responsible for data that is available, accurate, and secure. For certain classes of data, however, syntactically valid, but incorrect, data entered into a database can result in near-immediate, catastrophic financial losses for an organization. For example, in 2010, Apple Taiwan mispriced a Mac Mini at NTD 19,900 in its online store, when it intended to sell it for NTD 47,710. Over 41,000 customers purchased the machine at the offered price, leading to a loss in excess of 1 billion NTD (Anonymous, 2010). Criminals at a publicly traded company successfully falsified accounts for 16 years to extract USD 2.9 billion from shareholders (Drew, 2012). Historically, Nick Leeson caused the collapse of Barings, then Britain's oldest bank. He hid his illegal transactions using an "error account" - a financial account used normally as a stopgap to handle human accounting errors (Leeson & Whitley, 1996).

DOI: 10.4018/JDM.2016010102

These examples all illustrate a specific kind of low-incident data validation problem that can be particularly threatening to the well-being of an organization. The data entered into the databases were syntactically legitimate. The deviancy of the data could only be detected by comparing statistical parameters of the data against statistical parameters of the data set to which the data belongs. However, such statistical analysis does not identify the data as being wrong, but rather as suspicious, requiring human intervention for detection. Furthermore, it is critical that the deviancy in the data be detected at the point of data entry, not during an audit that could occur weeks after the incident. Apple Computers lost revenue within minutes of the mis-keyed data entering the system. For the two fraudulent examples, damage occurred the moment the mis-keyed transactions were stored in the database.

These kinds of data validation problem are increasingly common (Staples, Zhu, & Grundy, 2016). For example, more organizations depend upon larger databases in the "big data" era (Chen, Chiang, & Storey, 2012), leading to "dangerous data" in the sense that crucial decision making can occur on incorrect or problematic data. The issue underlying these kinds of data validation problems is that the input data is syntactically correct. However, the input data deviates substantively from data already stored in the database. The specific issue, which we refer to as *part-whole validation,* is summarized as follows:

- The part-whole validation problem is a low incident, high impact one. As a result, many organizations often do not anticipate it.
- Input data that create the part-whole validation problem are syntactically valid. For example, eventually, prices of Mac Minis will drop to NTD 19,900. However, at the time, such prices were unusual enough that they should have been identified as being an anomaly.
- Standard database languages do not have any syntax to support part-whole validation, so there is no normal validation check embedded within an implemented database. Database developers do not normally spend time developing these types of validation checks which are difficult to program. The above examples illustrate a few of the cases where part-whole validation problems have arisen.
- Part-whole validation requires comparing input data against existing database records, making them potentially computationally expensive checks.

Part-whole validation, then, is a problem worthy of study. The specific scenario we are trying to address occurs when a database administrator is dealing with a normalized database. The administrator wants to establish certain part-whole validation rules on the database. The algorithms for handling part-whole validation are known (e.g., Chiang, Pell, & Seasholtz, 2003; Elahi, Li, Nisar, Lu, & Wang, 2008; Georgiadis, et al., 2013; Gupta, Gao, Aggarwal, & Han, 2014). However, implementing all the part-whole validation rules in a modern relational database would require substantial work. Specifically, the database developer would be required to:

1. Create views to represent the data to be validated. Such data can arise from multiple tables.
2. Create triggers on the views. These triggers must be written in a procedural language and three new triggers created for every part-whole validation rule desired -- one to insert, update, and delete data.
3. Create the metadata upon which the triggers rely.

Instead, it would be preferable for the techniques of part-whole validation to be embedded as part of the declarative database language. As an analogy, one can manipulate relational tables with a standard procedural language, but it is more cognitively efficient to have SQL on top of the procedural language. An SQL instruction translates to the procedure, with the administrator oblivious to how this occurs. However, proposing such a language (or, in our case, language extension) requires, not