Requirements elicitation has been reported to be the stage of software development when errors have the most expensive consequences. Users usually find it difficult to articulate a consistent and complete set of requirements at the beginning of a development project. Prototyping is considered a powerful technique to ease this problem by exposing a partial implementation of the software system to the user, who can then identify required modifications. When prototyping data-intensive applications a so-called prototype database is needed.

This paper investigates how a prototype database can be built. Two different approaches are analysed, namely test databases and sample databases; the former populates the resulting database with synthetic values, while the latter uses data values from an existing database. The application areas that require prototype databases, in addition to requirements analysis, are also identified. The paper reports on existing research into the construction of both types of prototype databases, and indicates to which type of application area each is best suited. This paper advocates for the use of sample databases when an operational database is available, as is commonly the case in software maintenance and evolution. Domain-relevant data values and integrity constraints will produce a prototype database which will support the information system development process better than synthetic data. The process of extracting a sample database is also investigated. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: database sampling; software prototyping; software requirements engineering; data-intensive applications

1. INTRODUCTION

Software prototyping is a risk-reduction technique commonly used in software development when the requirements of the system under development are not well defined. For the purposes of this paper,

*Correspondence to: Jesús Bisbal, Department of Computer Science, Trinity College Dublin, Dublin, Ireland.
†E-mail: jesus.bisbal@cs.tcd.ie
‡In general, a prototype is a model of the software to be built which exhibits the desired properties of the final product [1]. Numerous techniques have been developed to create such models (e.g. simulations, reusable components, high-level languages, breadboards), not all of which result in an implementation of the system.
a software prototype is a partial implementation of the system to be built. This software prototype is exposed to user comment and then refined to meet additional requirements or correct errors. This process is repeated until an adequate prototype has been developed, resulting in a better understanding of the requirements for this part of the system. Whether software prototyping is to be used as the main software development paradigm or as part of a wider framework has been widely discussed elsewhere [2–4] and is out of the scope of this paper. Prototyping is, nonetheless, a technique applicable to a wide range of software development projects, which ultimately leads to more usable software.

This paper is concerned with the process of prototyping data-intensive applications, in order to support the development or maintenance of such applications. Thus in addition to the prototype application, a prototype database is also required. It has been recognized [4,5] that, wherever possible, operational data as opposed to synthetic data should be used to populate prototype databases. However, there has been little research into the process of creating or generating such a prototype database from operational data. This paper analyses the database prototyping process and, in particular, database sampling, that is, the construction of prototype databases populated with operational data. At the simplest level, some applications of sampling are only concerned with the origin of the data, namely that it should be operational data. However, many applications require the resulting database to contain semantic information similar to that of the operational database (see Section 4). Thus in consistent database sampling the objective is to select data items so that the resulting sample database satisfies predefined criteria, generally a set of integrity constraints. Using the entire operational database for prototyping purposes only is likely to be too costly and unnecessary.

It has been reported that as much as 60% of total software development costs are devoted to enhancing existing applications, to add or modify functionality, rather than developing new applications [6]. Thus in at least 60% of projects it is reasonable to expect that an operational database exists from which the sample data can be extracted. Legacy migration and Web-enabling existing applications are examples of projects in which a database would be available. The research presented here was initially motivated in the context of legacy information systems migration [7], where the need to identify a sample of an existing operational database is essential to the success of any migration project [8] (see Section 4.2.2).

The reminder of this paper is organized as follows. The next section introduces the terminology to be used, comparing database prototyping and database sampling. Section 3 reviews the existing work related to database prototyping. In Section 4 the application areas of database prototyping will be discussed, in a wider context beyond that of software prototyping. A framework to evaluate database prototyping approaches is presented in Section 5. Section 6 describes the process of database sampling, with a focus on the consistency of the resulting database. The final section summarizes the paper and gives a number of future directions for this research.

2. DATABASE PROTOTYPING AND DATABASE SAMPLING

For the purposes of this paper the term prototype database is defined as follows.

Prototype database: any database used to model (part of) the data and/or the semantics of another database.
Note that this definition is not concerned with how a prototype database is built, only to what it is used for.

This paper proposes that a prototype database should be populated using representative data from an operational database without having to use the entire database, which could be very large. The resulting prototype database is called a sample database.

**Sample database**: a prototype database populated with data from an existing database, according to predefined data selection criteria.

Data can be sampled from a database according to many different criteria. For example, data items to be included in the sample database could be randomly selected from the operational database. Frequently, however, it is important to include semantic information, for example satisfying a set of integrity constraints, in the sample database. The resulting sample database is referred to as a consistent sample database.

**Consistent sample database**: a sample database where the data selection is performed following a predefined set of criteria which are used to evaluate the consistency of the resulting database.

Finally, when it is not possible or necessary to use data from an existing database the resulting prototype database is called a test database.

**Test database**: a prototype database which is populated using synthetic values.

A prototype database can, in fact, have elements of both a test database and a sample database. This would be the case, for example, if a subset of the data of an operational database (e.g. the set of first names and last names of employees) is used as the domain for the values of the attributes in a prototype database. The resulting database can be seen as a sample database because each individual data value has been taken from an existing database. It can also be seen as a test database, since the database as a whole is not a direct result of sampling an existing database (e.g. if first and last names are combined randomly, the resulting full names may not correspond to any existing name in the operational database). This may be an appropriate approach in the case of an operational database containing variable quality data, as would be common when dealing with legacy databases [8].

With this approach, the resulting prototype database could be of better quality than its source database, while at the same time containing data values with which the users would be familiar. However, if the focus is on achieving high similarity with the original database (including its semantics), a common requirement in many prototype database applications, then this prototype database construction method would not be appropriate.

Another mixed approach to database prototyping which could not be regarded exactly as producing either test databases or sample databases would result if part of the prototype database is populated with data from an existing database, and another part using synthetic values.

For simplicity, all these mixed approaches to database prototyping are also referred to as test databases in this paper, because the focus here is on building consistent sample databases as defined above.

From the discussion given in this section, the terms test database and sample database can be seen as two extremes of a spectrum of possible types of prototype databases, depending on how they have been populated. This view is exploited in Section 5 when describing a framework for evaluating prototype database constructions methods.
3. RELATED WORK

Most of the existing approaches to database prototyping generate test databases, that is, they populate the database with data values not necessarily related to the application domain at hand (e.g. ‘name1’, ‘name2’, . . . as values for an attribute representing persons names). In the context of the data-intensive applications prototyping, existing approaches can be classified into those for database performance evaluation and those for requirements analysis. The main goal of the former is to generate large amounts of data, without including specific semantics. The objective of the latter is to produce a database which is highly similar to the database it models and therefore they focus on the semantic contents of the resulting database.

Gray et al. [9] report on a database prototyping method for database performance evaluation. His paper describes sequential and parallel algorithms to populate a database with large amounts of data. The data values are strictly randomly generated using several probability distributions. The approach presented in [10] is also concerned with database performance evaluation. It shows that generic database benchmarking tests prove unsatisfactory for high-performance databases, because their underlying schemes are too simplistic and the data volumes being considered are too small.§ Bates et al. [10] paper proposed an alternative, more realistic, benchmark model suitable for large parallel databases. It describes a toolkit used to generate database prototypes based on more realistic requirements, in terms of workload, semantic information, and data values. The domain values for some attributes are taken from predefined domains (e.g. files with female names, male names, and family names), thus following one of the mixed approaches to database prototyping outlined in Section 2.

Some approaches have been proposed to build test databases with significant semantic information, always in terms of the set of integrity constraints being satisfied by the resulting database. The general mechanism for test data generation involves inserting data values into the database and then testing-and-repairing this data by adding/deleting data items so that it eventually meets the specified constraints. Most solutions proposed for test data generation deal with a very reduced set of constraint types, typically functional dependencies, referential integrity constraints, or inclusion dependencies. Noble [11] describes one such method, considering referential integrity constraints and functional dependencies, and populating the database mainly with synthetic values, although the user can also enter a list of values to be used as the domain for an attribute.

A notable exception is found in [12] where a subset of first-order-logic (FOL) is used to define the set of constraints that the generated test data must meet, thus allowing complex constraints to be defined.

A different database prototyping approach is presented in [13]. This method first checks for the consistency of an extended entity-relationship (EER) schema defined for the database being prototyped, considering cardinality constraints only. Once the design has been proved consistent, a test database is generated. To guide the generation process, a so-called general dependency graph is created from the EER diagram. This graph represents the set of referential integrity constraints that must hold in the database and which is used to define a partial order between the entities of the EER diagram. The test data generation process populates the database entities following this partial order. Löhr-Richter and

---

§Benchmarks have been updated since this paper was published. The latest version of the benchmark referred to, TPC-C version 5 at http://www.tpc.org/, is effective since 26 February 2001. This new version increases the data volumes according to technological advances. However, the business model still enforces a significantly simpler set of integrity constraints than that used in [10], so the claim of excessive simplicity made by Bates et al. can still be considered to hold.
Zamperoni [14] further developed the test data generation step used by this method, and recognized the need for additional information in order to generate relevant data for the application domain.

Tucherman et al. [15] present a database design tool which prototypes a database, also based on an entity-relationship schema. The tool automatically maps this design into a normalized relational schema. Special attention is paid in this contribution to what are called restrict/propagate rules; that is, to enforcing referential integrity constraints. When an operation is going to violate one such constraint, the system can be instructed to either block (restrict) the operation so that such violation does not occur or to propagate it to the associated tables by deleting or inserting tuples as required. No explicit reference is made as to how the resulting database would be populated for prototyping, although it identifies the possibility of interacting with the user when new insertions are required, as the user may be the only source for appropriate domain-relevant data values.

Mannila and Räihä [16] described a mechanism for efficiently populating a database relation so that it satisfies exactly a predefined set of functional dependencies. Such a relation can be used to assist the database designer in identifying the appropriate set of functional dependencies which the database must satisfy. Since this relation satisfies all the required dependencies and no other dependency, it can be seen as an alternative representation for the dependencies themselves. A relation generated using this method is expected to expose missing or undesirable functional dependencies, and the designer can use it to iteratively refine the database design until it contains only the required dependencies.

All the database prototyping approaches described above populate the database using data values not related to the application domain. This limits their applicability, as will be described in Section 4.2. It must be noted that the need to build a prototype database by sampling operational data, as is advocated in this paper, arises in a context significantly different to that which motivated existing database prototyping approaches. Methods that produce synthetically generated databases are needed when developing completely new applications. Currently, there is an increasing need to develop or extend applications based on existing software and data (maintenance, migration, etc.) which motivates the need to address the traditional problem but with an additional constraint, the availability of existing applications and related data. This idea motivated an alternative line of work [17] originally proposed by Bisbal et al. [18]. The authors discussed a method for building sample databases, as opposed to test databases. The resulting prototype database is expected to support mainly requirements analysis and testing activities. The main objective is to include as much semantic information in the resulting database as possible, in order to build a prototype database highly similar to the database it models, referred to as the source database. This method relies on a graphical representation of integrity constraints, called an insertions chain graph, which represents different types of relationships between data items in the source database. This representation is then used to guide the sampling process, identifying which data items need to be inserted into the sample database in order for it to satisfy the specified set of integrity constraints. The same problem was addressed in [19], considering the satisfaction of sets of functional dependencies. This method was then generalized in [20], abstracting the sampling process from the specific data model (e.g. relational, object-oriented, hierarchical, semi-structured) to be sampled, and the set of integrity constraints to be satisfied by the resulting sample.

4. APPLICATIONS OF DATABASE PROTOTYPING

Software prototyping, as described in Section 1, is a technique closely associated with user requirements analysis. Although this is the main motivation for prototyping, other application areas...
have also been identified, including user training and software testing [3]. In addition, database prototyping, and in particular database sampling, is also applied to other areas outside software prototyping, including data mining [21], and approximate query evaluation [22,23]. This section analyses the application areas that require the construction of database prototypes. They are classified in terms of which type of database prototype, namely test database or sample database, can be used in each type of application. Figure 1 summarizes the different application areas described here.

4.1. Applications of test databases

Database prototyping is commonly used to support several stages of the information systems development process. These applications of database prototyping can, to some extent, be supported using synthetic data, that is, test databases. Other applications of database prototyping are associated with the availability of operational data, as will be discussed in Section 4.2.

4.1.1. Information system development

All prototype database construction methods available in the literature, with the exception of the work derived from the research presented in this paper [17–20], populate the database using synthetic values
(see Section 3). Although sample databases support the information system development process better (refer to Section 4.2.1), there may not always be an operational database from which to sample. Therefore, a test database may be the only alternative.

When prototyping data-intensive applications, in order for it to be realistic, a prototype should ideally interact with a database with the same schema as the one it will have when in production. A test database may fulfill these requirements, depending on both its semantic contents and the semantic needs of the prototyping application.

A test database can support the following stages of the information system development process.

4.1.1.1. Requirements analysis. This is regarded as the most common motivation for prototyping an application [24]. As described in Section 1, a prototype is exposed to user comment to gain better understanding of user requirements. Providing the user with a realistic prototype, in terms of behaviour and look-and-feel, may be imperative in this context. When prototyping data-intensive applications, a prototype database is needed to model functional aspects of the application.

4.1.1.2. Database design. Experimenting with different design alternatives facilitates a better understanding of their completeness and correctness [11,12,15]. A partially implemented database, even if populated only with synthetic values, will support the evaluation of different design options, by exposing missing and undesirable relationships between data items.

4.1.1.3. Testing. At the latter stages of software testing, applications must be tested in conditions as similar as possible to those they will encounter during operation [5]. As outlined above, a test database could lead to a realistic prototyping environment. Examples of such stages of software testing include system functional test [25], performance evaluation [9], and back-to-back testing [3].

4.2. Applications of sample databases

An alternative to populating prototype databases with synthetic values is using data sampled from an existing operational database. The applications where a sample database would be helpful are analysed next.

4.2.1. Information system development

All the stages of the information system development process analysed in Section 4.1.1 can also be supported using sample databases. In fact, it is generally recognized [4,5] that test databases are not as useful and that sample databases, when available, should be used instead. Section 4.1.1 identified the need for a prototype to interact with a database with a schema as similar as possible to that of the production database. If, in addition to the same schema, a prototype database contains domain-relevant data values, this will make the prototype more realistic and hence improve its usefulness. In particular, requirement analysis and testing can be expected to be more effective if operational data is used instead of synthetic data. Users will be able to identify the data items the prototype is manipulating, understanding their semantics, and thereby providing more useful feedback to the developers. Also, testing will detect those errors that are more likely to occur during operation [5]. Additionally, user training can be particularly well supported by a sample database.
4.2.1.1. User training. Users should only be trained with domain-relevant data; synthetic data would be of very limited use. They need to be familiar with the actual data they will view when using the production application and, therefore, operational data should be used to populate a prototype database in this context.

4.2.2. Legacy information system migration

A particular type of information system development project is that of legacy information system migration. This can be defined as follows [7].

Legacy information system migration. Moving an information system to a more flexible environment which allows information systems to be easily maintained and adapted to new requirements, retaining original system data and functionality without having to completely redevelop them.

In this context operational data is certainly available for sampling, as it itself is a part of the system to be migrated. This process will require a prototype database to support the development of the target system, as described in Sections 4.1.1 and 4.2.1. Additionally, in this particular type of project, the data must be migrated from the legacy environment into the target environment. This is a crucial part of any migration project, and some methodologies have been proposed to support this process [8,26]. The actual mapping process, from the legacy to the target database schemes, must be developed and tested thoroughly. In this testing process, a prototype database as similar as possible to the operational database, e.g. a sample database, will be invaluable. Using the entire database for testing purposes would probably be too costly.

5. EVALUATING PROTOTYPE DATABASE CONSTRUCTION METHODS

When the need for constructing a prototype database arises, it is necessary to identify the appropriate method to use. The requirements of the database prototyping application (e.g. the application areas described in Section 4) will determine the most suitable method(s). Existing approaches can be classified according to two orthogonal criteria: (1) the origin of the data used to populate the prototype database; and (2) the amount of semantic information the prototype database contains. Based on these two concepts, it is possible to select the appropriate method for constructing the prototype database for the application at hand.

Synthetic versus operational data. This issue has already been discussed in Section 2.

Poor versus rich semantics. Existing solutions enforce different sets of integrity constraints in the resulting prototype database, as analysed in Section 3. Random values generation [9] and random sampling [21], for example, would result in databases with poor semantic information as no particular constraints would be enforced. An approach which includes a highly expressive language (e.g. FOL) used to define the set of integrity constraints being enforced would produce databases with richer semantic information.

The kind of support that a prototype database built using a particular method provides to the information system development process can be assessed by identifying where this method falls according to the two classifying criteria given above. These criteria can, therefore, be seen as a
framework for the evaluation of prototype database construction methods. Figure 2 shows a graphical representation of such a framework, where the two criteria, data origin and semantic content, have been displayed along each of the axes. Given one concrete prototype database construction method, the relative semantic richness it can enforce will provide a value for its Y coordinate in Figure 2. Similarly, the percentage \( \% \) of operational data used to populate a prototype database will define a value for its X coordinate. Figure 2 has been divided into four quadrants that indicate which methods are more appropriate for the various stages of the information system development process.

Methods in quadrant (1) (see, e.g., [20]) lead to databases highly similar to those the information system will use in production. For this reason the resulting prototype databases will be useful for user training, system functional testing and at the later stages of requirements analysis. Methods that fall in quadrant (3) (see, e.g., [9]) do not enforce complex integrity constraints and do not need to query other data sources to populate the constructing prototype database. Such methods can, therefore, efficiently generate large volumes of data, which makes them particularly appropriate for performance evaluation of information systems. Quadrant (2) represents a trade-off between efficiency (quadrant (3)) and faithfulness (quadrant (1)). For this reason methods in this quadrant (see, e.g., [12]) are appropriate for initial database requirements analysis where alternative designs must be explored (semantic information would be useful) and, therefore, several prototype databases may need to be generated (need for efficiency). Finally, methods in quadrant (4) (e.g. [21–23]) would not generally be useful to support information systems development. This quadrant indicates the use of operational \( \% \)A method could combine operational with synthetic data to populate a prototype database, as discussed in Section 2.
data and the inclusion of little semantic information in the prototype database being populated. Using operational data results in prototype databases very similar to the production database; not including semantic information leads to databases less similar to the operational database, which defeats the use of operational data within information system development. Methods within this quadrant are, however, the only ones used in applications of sampling such as data mining [21] and approximate query evaluation [23].

The above discussion places sample databases and test databases in the context of information systems development. Sample databases are those produced by methods that fall in quadrants (1) and (4), and test databases by those in quadrants (2) and (3). This paper is concerned with the construction of consistent sample databases, which result when using methods in quadrant (1), shaded in Figure 2. It is clear from this figure that this class of prototype databases can be used to support more stages of the development process than those that result from using methods in other quadrants. Therefore consistent sample databases are the preferred prototype database type to be used when operational data is available.

6. CONSISTENT DATABASE SAMPLING PROCESS

It should be clear after the discussion of the Sections 4 and 5 that, in general, prototype databases populated with operational data support all applications of database prototyping better than those populated with synthetic data. This section outlines the main issues involved when a consistent sample is to be extracted from a database.

As with any other process, the consistent sampling process can be described in terms of its inputs and its outputs, as shown in Figure 3. In this case, the inputs are the operational database to be sampled together with a set of database consistency criteria used to evaluate the output, that is, the resulting sample database. At the core of the sampling process lies the fact that the insertion of a data item into the sample database may also require, according to the consistency criteria, some other data items to be inserted. This in turn may lead to further new data items being inserted. This process will be repeated until the resulting database, as a whole, satisfies the input consistency criteria.

An initial implementation of the consistent database sampling process was reported in [18]. Consistency in this case was achieved using a so-called insertions chain graph, as outlined in Section 3.
A more general study of this process is given in [20], together with the design of a prototype of a consistent database sampling tool. This prototype can be extended to include additional consistency criteria not yet envisaged. The design principles needed to extend this tool are also analysed in that paper.

At this point the possibility that the source database itself may not be consistent with the set of consistency criteria should also be considered. This is indeed commonly the case with legacy databases. The process described above views the operational database as being composed by several independent and consistent portions of data, each one resulting from an initial insertion followed by a consistent completion. Even if the source database as a whole was not consistent with the input set of consistency criteria, one consistent subset could still exist and be used as a sample database (Section 7 discusses this topic further).

7. SUMMARY AND FUTURE WORK

This paper has analysed the role of database prototyping within software prototyping (testing, requirements analysis, design experimentation, and user training) and in the context of other types of applications (data mining, approximate query evaluation). Two different approaches to database sampling have been identified, namely test databases and sample databases, depending on the origin of the data used to populate the resulting prototype database, operational or synthetic respectively. A framework for evaluation of which approach is more appropriate for different classes of applications has been presented. It was shown that, when possible, a sample database should be used as it results in prototype databases more similar to the database it models. The argument is a qualitative one, as users will view familiar data being manipulated by the application, testing is more likely to detect errors that will occur during actual operation, etc. Quantitatively comparing the sample database and the test database could prove difficult; experimentation is likely to be the only way of proving that one method is superior to the other.

Future work is currently being focused on studying the consistent database sampling process. Formal semantics will provide a better understanding of this process, as well as the extent to which sample databases can be used to model operational databases. This semantics will also provide a benchmark for consistent database sampling tools.

Consistent database sampling in the case of an inconsistent source database also requires further research. In particular, the use of consistent database sampling as a data cleaning mechanism will be the focus of this line of work.

REFERENCES


**AUTHORS’ BIOGRAPHIES**

**Jesús Bisbal** is currently a software analyst with a multinational IT services company, and also a part-time lecturer with the Pompeu Fabra University, Spain. Bisbal received his computer engineering degree at the Technical University of Catalonia, Spain, and he holds a PhD in computer science from Trinity College Dublin, Ireland. His research interests include database theory, health informatics, data mining and legacy systems migration.
Jane Grimson is Vice-Provost and Director of the Centre for Health Informatics, Trinity College, in Dublin, Ireland. Her research interests include distributed database systems and health informatics. She holds a bachelor’s degree in computer engineering from Trinity College Dublin, and a master’s and doctorate in computer science from the University of Toronto and the University of Edinburgh, respectively. She is a Chartered Engineer, a member of the IEEE and the ACM, fellow and past president of the Institution of Engineers of Ireland, president of the Irish Computer Society, the Health Informatics Society of Ireland, and the Irish Academy of Engineering, and a fellow of the Royal Academy of Medicine of Ireland, and the British Computer Society.