Database testing – an overview

Augustin-Iulian Ionescu*

**Department of Computers and Information Technology*
(e-mail:ionescu.augustin@cs.ucv.ro)
University of Craiova, Decebal Blv. No. 107

Abstract: The differences between data and information imply different approach of data quality and information quality. Sometimes apparently correct data can generate wrong information because the information represents an interpretation of the data in a given context and the interpretation is a very subjective process. Using the PSP/IQ model, this paper emphasizes the most important metrics for data information testing and attempts to answer what database testing means. Database quality analysis raises some specific issues that require special approaches. A very well designed database can be useless if the data timeliness is not permanently supervised but also, correct data can be useless if the database structure does not allow their correct association. This paper aims to answer the question: what does "testing a database" means.

Keywords: database, information, metrics, PSP/IQ model, quality, testing

1. INTRODUCTION

The low quality of data/information has a major negative impact over the hall activity into an organization. Because the data set at the fundament of the data valued chain, we can conclude that every error that affect the data life cycle can affect the derivate information and thus the decision based on this information [Ionescu, 2009]. In consequence all organizations are concerned to assure a high quality of their databases. All related activities must eliminate the dirty data not only useless but even dangerous for the quality of decisions taken at executive level.

2. PSP/IQ MODEL

There are four possibilities to approach the *quality* concept :

- the quality is *excellence*;
- the quality is *value*;
- the quality is *fitness for requirements*;
- the quality is *fitness for user's expectations.*

The model *PSP/IQ* (Product and Service Quality for Information Quality) was presented in (Beverly, 2002). This model (Table 1) can easily be put in touch with a set of metrics based on the fitness for specification as on the fitness for user's expectations.

The metrics proposed by the authors of this model have the following meanings:

- **accessibility** - the extent to which a data/information is available to as many people as possible or can be quick retrieved;
- **accuracy** - the extent to which the information is correct and safe for use;
- **actuality (timeliness)** – the extent to witch the information reflects reality at a moment clause enough to the moment of the information use;
- **added value** - the extent to which the information produce benefits by it using or distributing;
- **appropriate amount of information** – the extent to witch the available amount of information is sufficient to resolve the user's problems.
- **credibility** – extent that the information can be considered true or at least credible;
- **completeness** - the extent to which the information is sufficient to solve current tasks;
- **concise representation** - the extent to which information is represented in a compact, no redundant form;
- **consistent representation** - the extent to which representation is unambiguous and contains no logical inconsistency;

Table 1. PSP/IQ model

	Satisfaction of specifications	Meet and exceed of consumer expectations
Product Quality	**Sound Information** The characteristics of the information supplied meet IQ standards.	**Useful Information** The information supplied meets information consumer task needs.
Service Quality	**Dependable Information** The process of converting data into information meets standards.	**Usable Information** The process of converting data into information exceeds information consumer needs.

- **intelligibility** - the extent to which information can be understood by different categories of users;
- **interoperability** - the extent to which information from different sources is represented in the same format, in appropriate language, with appropriate symbols and units of measurement;
- **objectivity** - the extent to which information is generated by some unbiased and repeatable processes and measurements are made with authorized procedures;
- **relevance -** the extent to which the information is useful for solving specified tasks;
- **reputation** - the extent to which information is generated by an authorized and esteemed source;
- **safety** - the extent to which access to information is restricted so that it can not be intentionally destroyed or used for deceptive purposes;
- **usability** - extent that the information is easy to manipulate in various applications.

The relationship between the PSP/IQ model and the information metrics presented above are summarized in Table 2.

Table 2. The relationship between PSP/IQ model and the information metrics (Beverly, 2002)

	Satisfaction of specifications	**Meet and exceed of consumer expectations**
Product Quality	**Sound Information** - concise repre-sentation; - completeness; - consistent representation.	**Useful Information** - appropriate amount of information; - relevance; - inteligibility - interoperability; - objectivity.
Service Quality	**Dependable Information** - actuality; - security;	**Usable Information** - added value; - credibility; - accessability; - reputation; - reputation.

3. DATABASE QUALITY ISSUES

As shown in (Ionescu, 2008), in the study of data quality / information some specific issues related to accuracy and dynamics can be emphasized.

Especially when working with data about people, data accuracy is very difficult to prove. For example the confusion between "valid date" and "true date" is frequently observed.

Those who collect and enter data into the computer basically have very few ways to check if a document is official or falsified. Such research involves specialists, special equipment and high costs.

Even if those who declare certain data are in good faith, they may occur as false. For example, in certain social settings or in certain geographic area, date of birth is known only approximately, in relation to certain special events or family tradition.

The dynamic nature of the data makes accurate data at some moment to become false sometimes later. For example, address, phone number, email address are submitted to relatively frequent changes over time and in general people do not immediately notice these changes or not declare them at all.

Some changes, such as the change of name by marriage, require the implementation of some means of tracking all the time-related changes**.** For example, it is possible for the same person to publish articles under various names, which adds complexity to the verification of statements from CV.

As transforming data into information is a matter of interpretation, sometimes accurate data does not represent accurate information. This happens frequently in Medicine where the same symptoms can lead to different diagnoses depending on the experience of the person who makes the interpretation.

4. WHAT SHOULD BE TESTED IN A RELATIONAL DATABASE

There are still people who confuse the database testing with the testing of the applications developed over this database. In reality the testing of a database represents a very complex and laborious process which include:

- Verification of accuracy of requirements (static testing, inspection);
- Verification of correctness of the capture of specific data integrity rules (static testing);
- Verification of correctness of the conceptual model (static test);
- Verification of correctness of the logical models (static test);
- Checking the correctness of transformation of the conceptual model in relational model (static test). This also involves the final structure optimization by normalizing data and eliminating duplicates, synonyms and homonyms (static testing);
- Verification of correctness of scripts intended to create the database and tables (static and dynamic testing);
- Verification of correct implementation of declarative integrity rules (dynamic testing);
- Checking the possibility of using national alphabets with diacritics (dynamic testing);
- Verification of correctness of relationships between tables (dynamic testing).
- Verification of correctness of the views (static and dynamic testing).
- Verification of implementation of stored procedures (static and dynamic testing).

- Verification of implementation of user functions (static and dynamic testing).
- Verification of correct implementation of triggers (dynamic testing).
- *Observation*! Currently stored procedures, user defined functions and triggers are fundamental elements of the database, with tables and views;
- Verification of correct implementation of secure access to database content (static and dynamic testing);
- Depending on load, performance measurement to assess the quality indexes created (dynamic testing);
- Realization of stress tests for large volumes of data and / or a large number of users to concurrently access (dynamic testing);
- Checking of the users rights grant;
- Cheking of the data timeliness.

Each type of the above tests involves several processes and must be done by a certain category of professionals.

Creating conceptual model is an iterative and laborious process, based on the requirements of the user. Obtained structure can be permanent modified based on discussions between designer and user.

Such conceptual model quality testing is performed by specialists in conceptual design model while relational model quality is checked by specialists SQL or database administrators.

One of the key roles of any information system is to enforce the business rules set by the owning organization. Time is the most significant barrier to effectively testing the implementation of a set of business rules (David, 2005). We can distinguish two component of time spend for the business rules testing:

- the time required to run sufficient tests to verify each of the rules;
- the time needed for testers to design test cases that not only exercise the functionalty of the system but also verify that a set of business rules have not been violated.

The information systems will need to enforce a lot of different business rules. The implementations of this rules is spread across the system. The type of implementation will also vary between rules:

- implementation in a declarative manner;
- implementation as a trigger;
- implementation in a rule repository;
- implementation in the application's source code.

Additionaly, business rules evolve freqquently, in line with changing business focus, organization oportunities and statuary regulations. Sometimes for diffent periods must be implemented different rules; this situation change a declarative rule in a more complicated time depending rule wich need a trigger or a rule repository for implementation.

To test a business rule one or more SQL queries must be generated. The automation of SQL queries generation is not a simple process.

It seems that software tools that can automatically generate all the necesary test cases for such situations do not exist yet.

Because many business rules are concerned with the organization, it is often difficult to formulate simple unit tests for them. Instead, integration testing must be performed, in wich several programs are executed in sequence to create the circumstances that might lead to a rule violation.

A special case is the testing of stored procedures, user functions or triggers because they can be implemented in various non-procedural programming languages, and can be tested at both the code (white-box testing) and functional level (black -box testing).

For example we consider a table tblPersoane created with the SQL statement:

```
CREATE TABLE tblPersoane(
  Marca char(10) NOT NULL,
  Categorie char(2) DEFAULT 'CD',
  CNP char(13) DEFAULT NULL,
  NumePersoana varchar(50) NOT NULL,
  InitialaPersoana   varchar(10)   NOT
NULL DEFAULT '-',
  PrenumePersoana   varchar(50)   NOT
NULL,
  Sex char(1) DEFAULT 'M',
  DataNastere date,
  Cetatenie   varchar(10)   NOT   NULL
DEFAULT 'română',
  Email varchar(40) DEFAULT NULL,
  Observatii varchar(40),
  CONSTRAINT  tblpersoane_pkey  PRIMARY
KEY (Marca),
  CONSTRAINT     tblpersoane_sex_check
CHECK (sex IN ('M ', 'F '))
  );
```

We also consider a trigger that checks the correctness of email addresses and normalizes the representation. The program, developed in pl/pgsql is presented in Fig. 1.

```
/*  trigger  for  e-mail  correctness
verification */

CREATE OR REPLACE FUNCTION
trVerificare_Email()
  RETURNS trigger AS
$BODY$
DECLARE
i integer;
k1 integer; k2 integer; k3 integer;
BEGIN
-- email must be represented only in
lower cases
NEW.Email = lower(btrim(NEW.Email));
-- verification of name correctness
  k1=0;k2=0;k3=0;
FOR i IN 1..length(NEW.Email) LOOP
```

```
  IF NOT ((substr(NEW.Email,i,1)
BETWEEN 'a' AND 'z') OR
        (substr(NEW.Email,i,1) BETWEEN
'0' AND '9') OR
        (substr(NEW.Email,i,1)='@') OR
        (substr(NEW.Email,i,1)='-') OR
        (substr(NEW.Email,i,1)='-') OR
        (substr(NEW.Email,i,1)='.'))
THEN
      RAISE EXCEPTION ' Email address
contains forbitten characters';
  END IF;

  IF (substr(NEW.Email,i,1)='@') then
k1=k1+1; END IF;
  IF (substr(NEW.Email,i,1)='.') then
k2=k2+1; END IF;

END LOOP;

  IF (k1!=1 OR k2!=1) THEN
    RAISE EXCEPTION 'Wrong format at
e-mail address;
  END IF;

RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;
```

Fig. 1. Trigger that checks the correctness of email addresses and normalizes the representation

Testing such a program can not be done directly but only by executing a set of instructions for *insert* or *update*.

Firstly a trigger based on the function from Fig. 1 was created, as follows:

```
DROP TRIGGER IF EXISTS
trVerificare_Email ON tblPersoane ;
CREATE TRIGGER trVerificare_Email
BEFORE INSERT OR UPDATE ON tblPersoane
FOR EACH ROW EXECUTE PROCEDURE
trVerificare_Email();
```

The first test will be an apparently simple one, especially if in the database's domain of use there are email addresses with length greater than that provided by the design of *tblPersoane* table.

```
INSERT INTO tblPersoane (Marca,
NumePersoana, InitialaPersoana,
PrenumePersoana, Email)
VALUES ('1234','Ionescu', 'F.',
'Augustin-Iulian',
'iulian1000@gmail.com');
```

Although this seemed to be a good test, the message received after execution is:

********** Error **********

```
ERROR: wrong code for SEX!!
SQL state: P0001
```

This message appears unrelated to the performed test, because it was not considered that before installing the analyzed trigger, over the same table another trigger was activated for testing **CNP** field and updating fields **Sex** and **DataNastere**. It follows that it is necessary to use another test to ensure correct not-null value for CNP, as follows:

```
INSERT INTO tblPersoane (Marca,CNP,
NumePersoana,
InitialaPersoana,PrenumePersoana,
Email)
VALUES ('1234','1520726163218',
'Ionescu', 'F.', 'Augustin-Iulian',
'augustin.iulian1000@gmail.com');
```

The message received after execution is:

ERROR: value too long for type character varying(20)

Note that, and this time the trigger proper testing was not possible. For not waiting the change of field Email description, the tester can use an appropriate correct email surrogate address like in following instruction:

```
INSERT INTO tblPersoane (Marca,CNP,
NumePersoana
,InitialaPersoana,PrenumePersoana,
Email)
VALUES ('1234', '1520726163218',
'Ionescu', 'F.', 'Augustin-Iulian',
'iai100@gmail.com');
```

The executed of insertion instructions allow the trigger testing, as follows:

```
DELETE FROM tblPersoane;
INSERT INTO tblPersoane (Marca, CNP,
NumePersoana, InitialaPersoana,
PrenumePersoana, Email)
INSERT INTO tblPersoane (Marca, CNP,
NumePersoana ,InitialaPersoana,
PrenumePersoana, Email)
VALUES ('1234', '1520726163218',
'Ionescu', 'F.', 'Augustin-Iulian', '
iai100@gmail.com');
DELETE FROM tblPersoane;
INSERT INTO tblPersoane (Marca, CNP,
NumePersoana , InitialaPersoana,
PrenumePersoana, Email)
VALUES ('1234', '1520726163218',
'Ionescu', 'F.', 'Augustin-Iulian',
'iai100@gmail.com.');
DELETE FROM tblPersoane;
INSERT INTO tblPersoane (Marca, CNP,
NumePersoana ,InitialaPersoana,
PrenumePersoana, Email)
VALUES ('1234', '1520726163218',
'Ionescu', 'F.', 'Augustin-Iulian',
'ia.i100@gmail.com');
DELETE FROM tblPersoane;
```

```
INSERT INTO tblPersoane (Marca, CNP, NumePersoana , InitialaPersoana, PrenumePersoana, Email)
VALUES('1234', '1520726163218', 'Ionescu', 'F.', 'Augustin-Iulian', 'iai100@@gmail.com');
```

Although the first two INSERT statement generate the same error messages, their meaning is different. The first message corresponds to a real case of wrong format (the last character is a point). The second message does not match a wrong format (the point can exist in the left or the right part of the email address) so put out an error in the operation of the program under test.

The last INSERT instruction do not generate any error message and this must be considered as an error in the program under test (the format of address is obviously wrong).

Many other test cases can be generated but one must see that even the very few cases analyzed previously can reveal the complexity of a trigger testing. This complexity derives from the fact that a trigger is oftently a program that performs various tests on the basis of sophisticated algorithms.

Verification of a query is a more difficult process than it seems at a first glance because it may involve decomposing the query into a set of relational algebra operators. Each of these operators must be tested independently. A special case is the testing of the impact of the number of instances from each table involved in the query on the time necesary to obtaine the result. In large databases such tests involve generating a large number of test data. On one hand these data have to be as close as possbile to real data and on the other hand they have to be anonymous, that is not to provide useful information to someone who has casually access to them. For example, the generation of the test CNP can be performed by enough sophisticated software, or more simply by taking some real CNP and assigning them to fictitious persons. For a few test cases, the trigger for the CNP validation can be used for the generation of test CNP. The trigger can have the following structure:

```
CREATE OR REPLACE FUNCTION trCNP_verif()
  RETURNS trigger AS
$BODY$
DECLARE
numar char(12);
suma bigint;
C smallint;
BEGIN
-- check for number of digits in CNP
IF length(NEW.CNP)<13 THEN
        RAISE EXCEPTION 'wrong CNP length!!';
END IF;
suma = 0;
-- CNP checksum calculation
numar='279146358279';
FOR k IN 1..12 LOOP
 suma= suma +CAST(substr(NEW.CNP,k,1) AS integer)*CAST(substr(numar,k,1) AS integer);
END LOOP;
       C = CAST(suma AS integer) % 11;
       IF C=10 THEN
C=1;
 END IF;
IF C <> CAST(substr(NEW.CNP,13,1) AS integer) THEN
   RAISE EXCEPTION 'wrong CNP code!!';
END IF;
-- substract the sex from CNP
 IF (substr(NEW.CNP,1,1)='1') OR
 substr(NEW.CNP,1,1)='5') THEN
 NEW.Sex='M';
ELSIF (substr(NEW.CNP,1,1)='2') OR
 (substr(NEW.CNP,1,1)='6') THEN
 NEW.Sex='F';
ELSE
RAISE EXCEPTION 'wrong code for sex!!';
END IF;
RETURN NEW;
END;
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER trCNP_verif BEFORE INSERT OR UPDATE ON tblPersoane
FOR EACH ROW EXECUTE PROCEDURE trCNP_verif();
```

Check a **view** is essentially checking the queries that underlie its creation. This involves the need to check the join conditions (if there are multiple tables) as well as the vertical and horizontal selection conditions and fairness aliases. A view test is often an integration test, involving the junction and/or union of more tables. Like any test integration, a multi-table view testing will not be conducted until all tests have been completed considered for each table.

Cheeking the data timeliness is possible only during the database maintenance and requires a rigorous plan checks usually performed by specialists of the organization that uses the database. Despite all efforts, ensuring accuracy of the data at any time is virtually impossible.

Great attention should be paid to the language-specific characters as their use depends on both the specifications in the database and setting of some parameters in the operating system.

In order to achieve certain performance and stress tests, it is necessary to create data libraries by automatically generating the dataset or by data adaptation of some old database.

5. CONCLUSIONS

Database testing is a very complex and time-consuming procedure which needs to be conducted on database's entire life cycle. The database testing is sometimes more difficult than the test of programs developed over it.

Special efforts should be done for the development of principles to guide databases testing and the implementation of the software tools for automatically generated test cases at least for the most common situations involving tests such as checking specific integrity rules implemented in a declarative manner or by triggers.

Testing database did not involve the creation of methods and techniques for testing out the classic scenarios (black-box, white-box and gray-box) (Shivani, 2012)). What is needed is to understand how to use these methods in the context of complex data structures.

REFERENCES

Beverly K. Kahn, Diane M. Strong (1998). *Product and Service Performance Model for Information Quality: An Update*, Information Quality Conference

Beverly K. Kahn, Diane M. Strong, Richard Y. Wang (2002). *Information quality benchmarks: product and service performance*, Communications of the ACM, Vol 45, No. 4ve

David Willmor, Suzanne M. Embury (2006). *Teting the implementation of business rules using intentional database testing*,
www.cs.man.ac.uk/~willmord/files/willmorembury-TAICPART06.pdf

Ionescu Augustin-Iulian, Dumitrascu Eugen (2008). *Continuous Data Quality Assurance*, Anale Universitatea din Craiova

Ionescu Augustin-Iulian, Dumitrascu Eugen, Enescu Nicolae-Iulian(2009). *Method of Testing the Quality of a Database During Exploitation*, MTC Greece

Shivani Acharya, Vidhi Pandya (2012). *Bridge between Black Box and White Box – Gray Box Testing Technique*, International Journal of Electronics and Computer Science Engineering, Vol 2-Nr. 1