CHAPTER FIVE

Data Warehouse Testing

Hajar Homayouni, Sudipto Ghosh, Indrakshi Ray
Colorado State University, Fort Collins, CO, United States

Contents

Abstract

Enterprises use data warehouses to accumulate data from multiple sources for data analysis and research. Since organizational decisions are often made based on the data stored in a data warehouse, all its components must be rigorously tested. Researchers have proposed a number of approaches and tools to test and evaluate different

ISSN 0065-2458
https://doi.org/10.1016/bs.adcom.2017.12.005

components of data warehouse systems. In this chapter, we present a comprehensive survey of data warehouse testing techniques. We define a classification framework that can categorize the existing testing approaches. We also discuss open problems and propose research directions.

1. INTRODUCTION

A data warehouse system gathers heterogeneous data from several sources and integrates them into a single data store [1]. Data warehouses are used for reporting and data analysis and form the core component of business intelligence (BI) [2]. The goal of data warehouses is to help researchers and data analyzers perform faster analysis and make better decisions [3]. Data warehousing also makes it possible to do data mining, which is the science of discovering patterns in the data for further decision-making, such as predictions or classifications [4]. Data warehouses often use large-scale (petabyte) data stores to keep archival as well as current data to enable data analyzers to find precise patterns based on long-term changes in the data.

Data warehouses are used in many application domains. A health data warehouse brings electronic health records from many hospitals into a single destination to help medical research on disease, drugs, and treatments. While each hospital focuses on transactions for current patients, the health data warehouse maintains historical data from multiple hospitals. This history often includes old patient records. The past records along with the new updates help medical researchers perform long-term data analysis. A weather data warehouse gathers observations from stations all around the world into a single data store to enable weather forecasting and climate change detection.

There are many components and processes involved in data warehousing. Fig. 1 shows the different components of a data warehouse system including (1) sources, (2) extract, transform, and load (ETL) process, (3) data warehouse, and (4) front-end applications.

The sources of a data warehousing system are data stores that provide data from various places. These sources store the data of entities belonging to one or more organizations. For example, the sources of a health data warehouse are obtained from multiple hospitals that are collaborating for medical research. There are different *models*, such as relational [5] or nonrelational [6] models, and *technologies*, such as database management systems

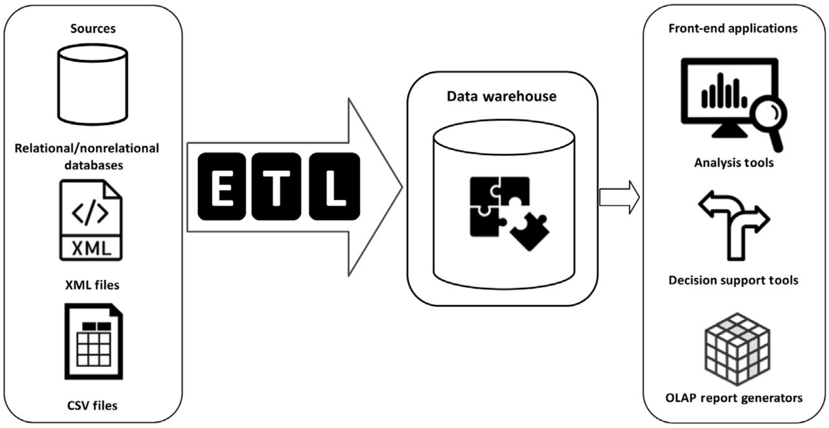

Fig. 1 Components of a data warehousing system.

(DBMSs) or extensible markup language (XML) or comma separated values (CSV) flat files that form the sources of data warehousing systems.

The ETL process selects data from the sources, resolves problems in the data, converts it into a common model appropriate for research and analysis, and writes it to the target data warehouse [1]. Among the four components presented in Fig. 1, the design and implementation of the ETL process require the largest effort in its development life cycle [3]. The ETL process presents many challenges, such as extracting data from multiple heterogeneous sources involving different models, detecting and fixing different types of errors in the data, and transforming the data into different formats that match the requirements of the target data warehouse.

The data warehouse keeps data gathered and integrated from different sources and stores a large number of records needed for long-term analysis. Implementations of data warehouses use various *data models*, such as dimensional or normalized models, and *technologies*, such as DBMS, data warehouse appliance (DWA), and cloud data warehouse appliance.

The front-end applications are in the form of desktop, web, and mobile applications that present business data with analysis to end users [3]. They include analysis and decision support tools and online analytical processing (OLAP) report generators. These applications make it easy for end users to construct complex queries to request information from data warehouses without requiring sophisticated programming skills.

Research is conducted and organizational decisions are made based on the data stored in a data warehouse [7]. For example, based on our health data warehouse, many critical studies such as the impacts of a specific medication on people from different groups are performed using patient, treatment, and medication data stored in the data warehouse. Thus, all the components of a data warehouse must be thoroughly tested using rigorous testing techniques. Although data warehouse design and implementation have received considerable attention in the literature, few systematic techniques have been developed for data warehouse testing [8].

There are a number of challenges in testing data warehouse systems:

- Heterogeneous sources and voluminous data involved in data warehousing make data warehouse testing harder than testing traditional software systems [7]. A comprehensive testing approach should take into account all possible sources and test inputs for adequate testing.
- Due to the confidentiality of data in the sources of data warehousing systems, testers typically do not have access to the real data. As a result, there is a requirement to create fake data with the relevant characteristics of real data that enables adequate testing.
- Most data warehouse testing approaches are created for a specific context and cannot be used for testing other data warehouse systems. This problem arises because testing approaches are typically designed based on business domain requirements and the data warehouse architecture. These testing approaches cannot be generalized and reused in projects with different domain requirements [9] and architectures.
- Data warehouse testing requirements are not formally specified, which makes them hard to verify. The tester needs to bridge the gap between the informal specifications and the formality required for verification and validation techniques [10].

ElGamal [9] presented several data warehouse testing approaches, and evaluated and compared them to highlight their limitations. The survey reported the comparison based on *what* (referring to the testing type), *where* (implying the data warehousing stage where the testing is applied), and *when* (stating whether the test takes place before or after data warehouse delivery) in a three-dimensional matrix. In this matrix, the rows represent the *where*-dimension that takes four values, namely, *sources to data store*, *data store to data warehouse*, *data warehouse to data mart* (a subset of data warehouse), and *data mart to front-end applications*. The columns of the matrix represent the *what*-dimension that takes three values, namely, *schema related tests*, *data related tests*, and *operational related tests*. The third dimension of the matrix is the *when*-dimension that takes two values, namely, *before system delivery* and *after system delivery*. The survey compared 10 data warehouse testing approaches and concluded that none of them addressed all the *what*, *where*, and *when* categories, and there are some test types that are not addressed by any of these approaches. In the matrix, the component being tested and the testing type are often described together which makes it hard to understand the comparison matrix. For example, the entries *data model* and *requirement testing* both fall under the *what* dimension of the matrix, which results in ambiguities in the interpretation of the matrix.

Gao et al. [11] compared contemporary data warehouse testing tools for data validation in terms of their operating environment, supported data sources, data validation checks, and applied case studies. This survey compared commercial and open-source approaches that test the quality of the underlying data in the target data warehouse but did not consider approaches for testing the other data warehouse components that are shown in Fig. 1.

In this chapter, we present a comprehensive survey of existing testing and evaluation activities applied to the different components of data warehouses and discuss the specific challenges and open problems for each component. These approaches include both dynamic analysis as well as static evaluation and manual inspections. We provide a classification framework based on *what* is tested in terms of the data warehouse component to be verified, and *how* it is tested through categorizing the different testing and evaluation approaches. The survey is based on our direct experience with a health data warehouse, as well as from existing commercial and research efforts in developing data warehouse testing approaches. The rest of the chapter is organized as follows. Section 2 describes the components of a data warehouse. Section 3 presents a classification framework for testing data warehouse components. Sections 4–6 discuss existing approaches and their limitations for each testing activity. Finally, Section 7 concludes the chapter and outlines directions for future work.

2. DATA WAREHOUSE COMPONENTS

In this section, we describe the four components of a data warehousing system, which are (1) sources, (2) target data warehouse, (3) ETL process, and (4) front-end applications. We use the health data warehouse as a running example.

2.1 Sources and Target Data Warehouse

Sources in a data warehousing system store data belonging to one or more organizations for daily transactions or business purposes. The target data warehouse, on the other hand, stores large volumes of data for long-term analysis and mining purposes. Sources and target data warehouses can be designed and implemented using a variety of technologies including data models and data management systems.

A data model describes business terms and their relationships, often in a pictorial manner [12]. The following data models are typically used to design the source and target schemas:

- *Relational data model*: Such a model organizes data as collections of two-dimensional tables [5] with all the data represented in terms of tuples. The tables are *relations* of rows and columns, with a unique key for each row. Entity relationship (ER) diagrams [13] are generally used to design the relational data models.
- *Nonrelational data model*: Such a model organizes data without a structured mechanism to link data of different buckets (segments) [6]. These models use means other than the tables used in relational models. Instead, different data structures are used, such as graphs or documents. These models are typically used to organize extremely large datasets used for data mining because unlike the relational models, the nonrelational models do not have complex dependencies between their buckets.
- *Dimensional data model*: Such a model uses structures optimized for end-user queries and data warehousing tools. These structures include *fact* tables that keep measurements of a business process, and *dimension* tables that contain descriptive attributes [14]. Unlike relational models that minimize data redundancies and improve transaction processing, the dimensional model is intended to support and optimize queries. The dimensional models are more scalable than relational models because they eliminate the complex dependencies that exist between relational tables [15].

  The dimensional model can be represented by star or snowflake schemas [16] and is often used in designing data warehouses. These types of schemas are as follows:
  - *Star*: This type of schema has a fact table at the center. The table contains the keys to dimension tables. Each dimension includes a set of attributes and is represented via a one dimension table [17].
  - *Snowflake*: Unlike the star schema, the snowflake schema has normalized dimensions that are split into more than one dimension tables. The star schema is a special case of the snowflake schema with a single level hierarchy.

The sources and data warehouses use various data management systems to collect and organize their data. The following is a list of data management systems generally used to implement the source and target data stores.

- *Relational database management system* (*RDBMS*): An RDBMS is based on the relational data model that allows linking of information from different *tables*. A table must contain what is called a key or index, and other tables may refer to that key to create a link between their data [6]. RDBMSs typically use Structured Query Language (SQL) [18] and are appropriate to manage structured data. RDBMSs are able to handle queries and transactions that ensure efficient, correct, and robust data processing even in the presence of failures.
- *Nonrelational database management system*: A nonrelational DBMS is based on a nonrelational data model. The most popular nonrelational database is Not Only SQL (NoSQL) [6], which has many forms, such as document-based, graph-based, and object-based. A nonrelational DBMS is typically used to store and manage large volumes of unstructured data.
- *Big data management system*: Management systems for big data need to store and process large volumes of both structured and unstructured data. They incorporate technologies that are suited to managing nontransactional forms of data. A big data management system seamlessly incorporates relational and nonrelational database management systems.
- *Data warehouse appliance* (*DWA*): DWA was first proposed by Hinshaw [19] as an architecture suitable for data warehousing. DWAs are designed for high-speed analysis of large volumes of data. A DWA integrates database, server, storage, and analytics into an easy-to-manage system.
- *Cloud data warehouse appliance*: Cloud DWA is a data warehouse appliance that runs on a cloud computing platform. This appliance benefits from all the features provided by cloud computing, such as collecting and organizing all the data online, obtaining infinite computing resources on demand, and multiplexing workloads from different organizations [20].

Table 1 presents some of the available products used in managing the data in the sources and target data warehouses. The design and implementation of the databases in the sources are typically based on the organizational requirements, while those of the data warehouses are based on the requirements of data analyzers and researchers.

For example, the sources for a health data warehouse are databases in hospitals and clinic centers that keep patient, medication, and treatment information in several formats. Fig. 2 shows an example of possible sources in the health data warehouse. Hospital A uses a flat spreadsheet to keep records of

Table 1 Available Products for Managing Data in the Sources and Data Warehouses

Product Category	**Examples**
DBMS	Relational: MySQL [21], MS-SQL Server [22], PostgreSQL [23]
	Nonrelational: Accumulo [24], ArangoDB [25], MongoDB [26]
Big data management system	Apache Hadoop [27], Oracle [28]
Data warehouse appliance	IBM PureData System [29]
Cloud data warehouse	Google BigQuery [30], Amazon Redshift [31]

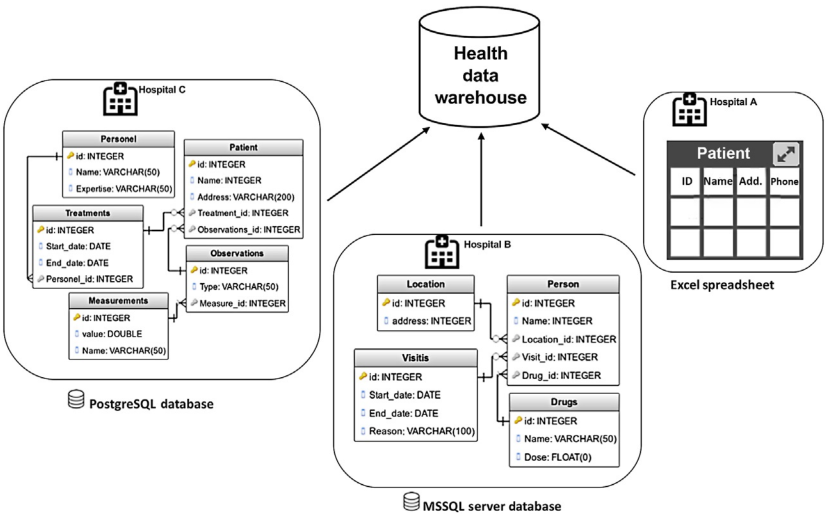

Fig. 2 Sample sources for a health data warehouse.

patient data. Hospital B uses an RDBMS for its data. Hospital C also uses an RDBMS but has a different schema than Hospital B. The data from different hospitals must be converted to a common model in the data warehouse.

The target data warehouse for health data may need to conform to a standard data model designed for electronic health records such as Observational Medical Outcomes Partnership (OMOP) Common Data Model (CDM) [32]. The OMOP CDM is a dimensional model that includes all the observational health data elements that are required for analysis use cases. The model supports the generation of reliable scientific evidence about disease, medications, and health outcomes.

2.2 Extract, Transform, and Load

The ETL process extracts data from sources, transforms it to a common model, and loads it to the target data warehouse. Fig. 3 shows the components involved in the ETL process, namely, extract, transform, and load.

1. *Extract*: This component retrieves data from heterogeneous sources that have different formats and converts the source data into a single format suitable for the transformation phase. Different procedural languages such as Transact-SQL or COBOL are required to query the source data. Most extraction approaches use Java Database Connectivity (JDBC) or Open Database Connectivity (ODBC) drivers to connect to sources that are in DBMS or flat file formats [33].

   Data extraction is performed in two phases. Full extraction is performed when the entire data is extracted for the first time. Incremental extraction happens when new or modified data are retrieved from the sources. Incremental extraction employs strategies such as log-based, trigger-based, or timestamp-based techniques to detect the newly added or modified data. In the log-based technique, the DBMS log files are used to find the newly added or modified data in the source databases. Trigger-based techniques create triggers on each source table to capture changed data. A trigger automatically executes when data is created or modified through a Data Manipulation Language (DML) event. Some database management systems use timestamp columns to specify the time and date that a given row was last modified. Using these columns, the timestamp-based technique can easily identify the latest data.

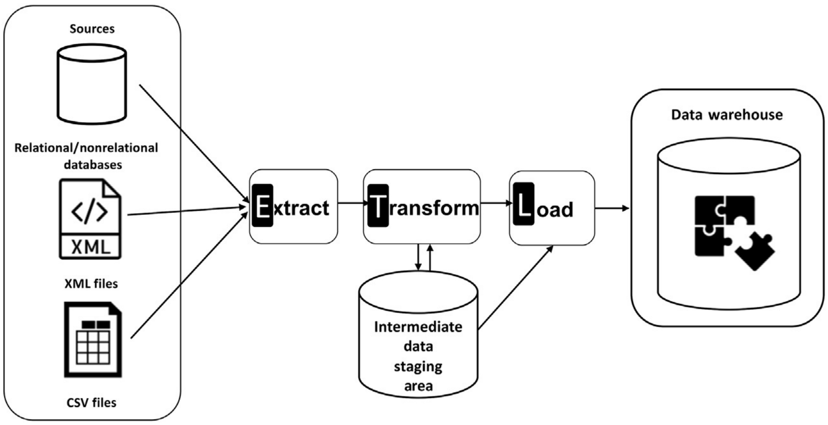
Sources
Relational/nonrelational databases
XML
XML files
CSV files
Extract
Transform
Load
Intermediate data staging area
Data warehouse

Fig. 3 General framework for ETL processes.

2. *Transform*: This component propagates data to an intermediate data staging area (DSA) where it is cleansed, reformatted, and integrated to suit the format of the model of a target data warehouse [3]. This component has two objectives.

   First, the transformation process cleans the data by identifying and fixing (or removing) the existing problems in the data and prepares the data for integration. The goal is to prevent the transformation of so-called dirty data [34, 35]. The data extracted from the sources is validated both syntactically and semantically to ensure that it is correct based on the source constraints. Data quality validation and data auditing approaches can be utilized in this step to detect the problems in the data. Data quality validation approaches apply quality rules to detect syntactic and semantic violations in the data. Data auditing approaches use statistical and database methods to detect anomalies and contradictions in the data [36]. In Table 2 we present some examples of data quality validation applied to data cleansing of patients in our health data warehouse.

   Second, it makes the data conform to the target format through the application of a set of transformation rules described in the source-to-target mapping documents provided by the data warehouse designers [33]. Table 3 presents examples of source-to-target mappings for generating a target table called *Patient* in our health data warehouse. The mappings include the names of the corresponding source and target tables, the source and target columns with their types, and selection conditions.
3. *Load*: This component writes the extracted and transformed data from the staging area to the target data warehouse [1]. The loading process varies widely based on the organizational requirements. Some data warehouses may overwrite existing data with new data on a daily, weekly, or

Table 2 Examples of Validation Applied to Data Cleansing

Validation	**Example of a Violation**
Incorrect value check	Birth_date=70045 is not a legal date format
Uniqueness violation check	Same SSN='123456789' presented for two people
Missing value check	Gender is null for some records
Wrong reference check	Referenced hospital=1002 does not exist
Value dependency violation check	Country='Germany' does not match zip code='77'

Table 3 Transforming Source Data to Generate Target Table *Patient*

Source			Target			
Table Name	**Column Name**	**Data Type**	**Table Name**	**Column Name**	**Data Type**	**Selection Condition**
PersonDim	PersonKey	Integer	Patient	Patient_id	Integer	Transform all the current patients
PersonDim	Name	String	Patient	Patient_name	String	Transform all the patients
PersonDim, AddressDim	AddressKey	Integer	Patient	Location_id	Integer	Transform all the patients with new addresses (after year 2000)
PersonDim, Concept	Sex	String	Patient	Gender	Integer	Transform all the patients with their sex using concept values female, male, other

monthly basis, while other data warehouses may keep the history of data by adding new data at regular intervals. The load component is often implemented using loading jobs that fully or incrementally transform data from DSA to the data warehouse. The full load transforms the entire data from the DSA, while the incremental load updates newly added or modified data to the data warehouse based on logs, triggers, or timestamps defined in the DSA.

The ETL components, namely, extract, transform, and load, are not independent tasks, and they need to be executed in the correct sequence for any given data. However, parallelization can be achieved if different components execute on distinct blocks of data. For example, in the incremental mode the different components can be executed simultaneously; the newly added data can be extracted from the sources while the previously extracted block of data is being transformed and loaded into the target data warehouse.

2.3 Front-End Applications

Front-end applications present the data to end users who perform analysis for the purpose of reporting, discovering patterns, predicting, or making complex decisions. These applications can be any of the following tools:

- *OLAP report generators*: These applications enable users and analysts to extract and access a wide variety of views of data for multidimensional analysis [37]. Unlike traditional relational reports that represent data in two-dimensional row and column format, OLAP report generators represent their aggregated data in a multidimensional structure called cube to facilitate the analysis of data from multiple perspectives [38]. OLAP supports complicated queries involving facts to be measured across different dimensions. For example, as Fig. 4 shows, an OLAP report can present a comparison of the number (fact) of cases reported for a disease (dimension) over years (dimension), in the same region (dimension).
- *Analysis and data mining*: These applications discover patterns in large datasets helping users and data analysts understand data to make better decisions [4]. These tools use various algorithms and techniques, such as classification and clustering, regression, neural networks, decision trees, nearest neighbor, and evolutionary algorithms for knowledge discovery from data warehouses. For example, clinical data mining techniques [39] are aimed at discovering knowledge from health data to extract valuable information, such as the probable causes of diseases, nature of progression, and drug effects.

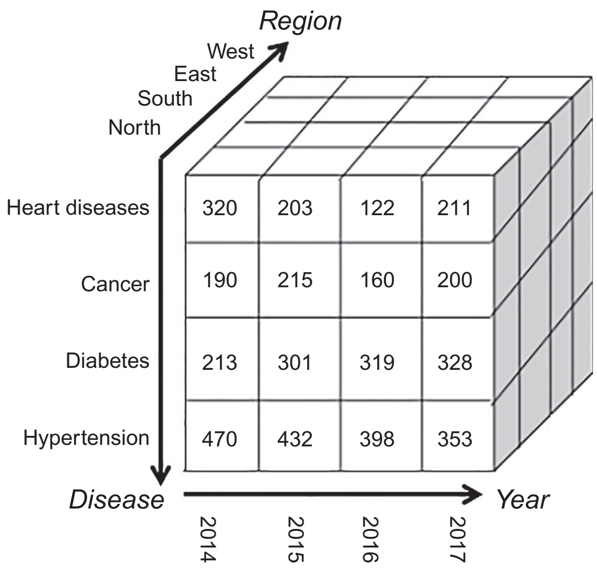

Fig. 4 OLAP cube example of the number of cases reported for diseases over time and regions.

- *Decision support*: These applications support the analysis involved in complex decision-making and problem solving processes [40] that involve sorting, ranking, or choosing from options. These tools typically use Artificial Intelligence techniques, such as knowledge base or machine learning to analyze the data. For example, a Clinical Decision Support [41] application provides alerts and reminders, clinical guidelines, patient data reports, and diagnostic support based on the clinical data.

3. TESTING DATA WAREHOUSE COMPONENTS

Systematic testing and evaluation techniques have been proposed by researchers and practitioners to verify each of the four components of a data warehouse to ensure that they perform as expected. We present a comprehensive survey by defining a classification framework for the testing and evaluation techniques applied to each of the four components.

Fig. 5 shows the classification framework for the techniques applicable to the sources, target data warehouse, ETL process, and front-end applications. The framework presents *what* is tested in terms of data warehouse components, and *how* they are tested. The following are the data warehouse components presented in the framework:

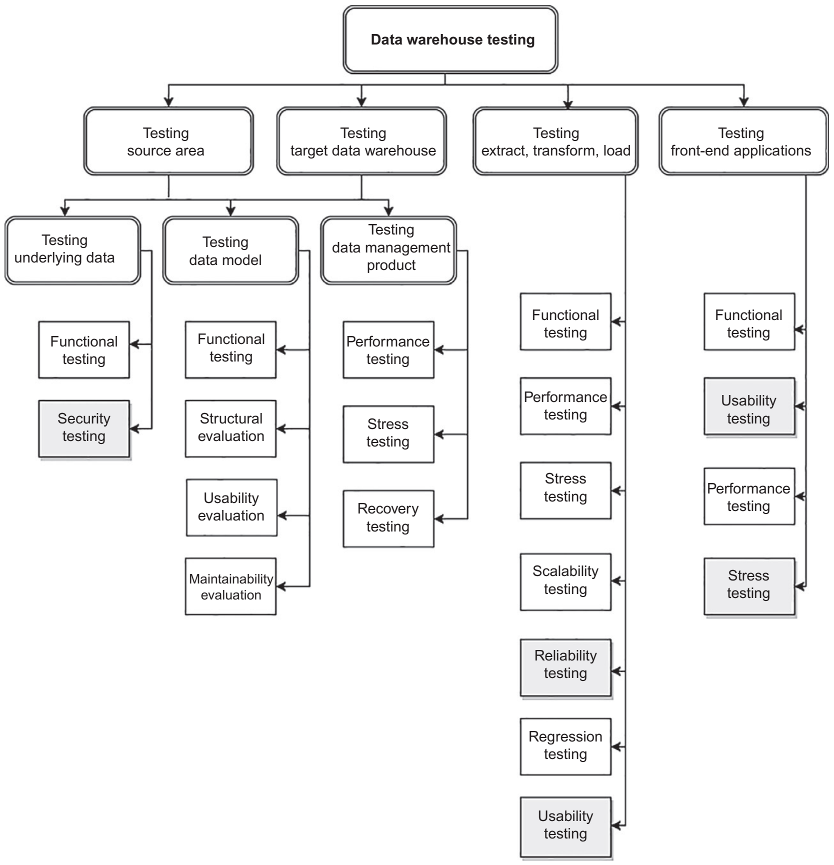

Fig. 5 Classification framework for data warehouse testing.

- *The sources and the target data warehouse* store data. As a result, the same types of testing and evaluation techniques apply to them. We consider three different aspects to classify the approaches used to test these two components; these are (1) testing the underlying data, (2) testing the data model, and (3) testing the product used to manage the data.
- *The ETL process* requires the largest effort in the data warehouse development life cycle [3]. As a result, most existing data warehouse testing and evaluation approaches focus on this process. Various functional and nonfunctional testing methods have been applied to test the ETL process because it directly affects the quality of data inside the data warehousing systems.
- *The front-end applications* in data warehousing systems provide an interface for users to help them interact with the back-end data store.

We categorize the existing testing and evaluation approaches as functional, structural, usability, maintainability, security, performance and stress, scalability, reliability, regression, and recovery testing. The shaded boxes represent the categories not covered by the existing testing approaches but that we feel are needed based on our experience with a real-world health data warehouse project.

Other researchers have also defined frameworks for testing techniques that are applicable to the different components of a data warehouse. Golfarelli and Rizzi [8] proposed a framework to describe and test the components in a data warehouse. They defined the data warehouse components as *schema*, *ETL*, *database*, and *front-end applications*. However, the *schema* and *database* are not exactly data warehouse components. Instead they are features of the sources and the target data warehouses. The framework uses seven different testing categories (functional, usability, performance, stress, recovery, security, and regression) applicable to each of the data warehouse components. Some nonfunctional testing techniques such as those for assessing scalability, reliability, and maintainability are not included.

Mathen [1] surveyed the process of data warehouse testing considering two testing aspects, which were (1) testing underlying data and (2) testing the data warehouse components. The paper focused on two components in the data warehouse architecture, i.e., the ETL process and the client applications, and discussed testing strategies relevant to these components. Performance, scalability, and regression testing categories were introduced. Although testing the sources and the target data warehouse is critical to ensuring the quality of the entire data warehouse system, they were ignored in Mathen's testing framework. Moreover, other functional and nonfunctional aspects of testing data warehouse components, such as security, usability, reliability, recovery, and maintainability testing, and existing methods and tools for each testing type were not included.

In Sections 4–6, we describe the testing and evaluation activities necessary for each component in detail and present the challenges and open problems.

4. TESTING SOURCE AREA AND TARGET DATA WAREHOUSE

In this section, we target the locations that store the data in a data warehousing system, namely, the sources and the target data warehouse. If problems exist in the sources, they should be resolved before the data is extracted and loaded into a target where fault localization is much more expensive [7]. Fault localization is the process of finding the location of faults in a software system. Due to the fact that there are many components and processes involved in data warehousing systems, if the faulty data are propagated to the target data warehouse, finding the location of the original fault that caused subsequent error states will require a lot of effort. As a result, testing the source area is critical to ensuring the quality of data being propagated to the target data warehouse.

The quality of the target storage area is also important [42] because this is the place where the data analyzers and researchers apply their queries either directly or through the front-end applications. Any problem in the target data warehouse results in incorrect information. Thus, testing must ensure that the target meets the specifications and constraints defined for the data warehouse.

We considered three different aspects to test the source area and the target data warehouse. These are (1) testing the underlying data, (2) testing the data model, and (3) testing the data management product.

4.1 Testing Underlying Data

In this testing activity, the data stored in the sources and the target data warehouse is validated against organizational requirements, which are provided by domain experts in the form of a set of rules and definitions for valid data. If the underlying data fails to meet the requirements, any knowledge derived from the data warehouse will be incorrect [43].

We describe existing functional and security testing approaches based on testing the underlying data in data warehouses as well as propose approaches based on our experience to achieve high quality data in a health data warehouse.

4.1.1 Functional Testing of Underlying Data

Functional testing of the underlying data is a type of data quality testing that validates the data based on quality rules extracted from business requirements documents. The data quality test cases are defined as a set of queries that verify whether the data follows the syntactic and semantic rules. This testing activity uses domain-specific rules, which are a set of business rules that are internal to an organization.

Examples of the data elements that are verified using data quality tests are as follows:

- *Data type*: A data type is a classification of the data that defines the operations that can be performed on the data and the way the values of the data can be stored [44]. The data type can be numeric, text, Boolean, or date-time; these are defined in different ways in different languages.
- *Data constraint*: A constraint is a restriction that is placed on the data to define the values the data can take. Primary key, foreign key, and not-null constraints are typical examples.

Examples of semantic properties that we suggest are as follows:

- *Data plausibility*: A restriction that is placed on the data to limit the possible values it can take. For example, a US zip code can only take five digit values.
- *Logical constraint*: A restriction defined for the logical relations between data. For example, the *zip code=33293* does not match the *country=Germany*.

The data quality rules are not formally specified in the business requirements. The tester needs to bridge the gap between informal specifications and formal quality rules. Table 4 presents some examples of informally defined data quality rules for electronic health records [45]. Table 5 shows test cases defined as queries to verify the data quality rules presented in Table 4. Assume that after executing the test cases (queries), the test results are stored in a table called *tbl_test_results*. In this table, each record describes the failed assertion. The record includes the *test_id* that indicates the query number, *status* that takes as values *error* and *warning*, and *description* that contains a brief message about the failure. An empty table indicates that all the assertions passed.

Table 4 Data Quality Rules for Electronic Health Records

	Field	**Data Quality Rule**	**Property**
1	Weight	Should not be negative	Semantic (data plausibility)
2	Weight	Should be a numeric value	Syntactic (data type)
3	Sex	Should be *male* or *female* or *other*	Semantic (data plausibility)
4	Sex	Should not be null	Syntactic (data constraint)
5	Start_date, End_date	Start_date of patient visit should be before End_date	Semantic (logical constraint)
6	Start_date, End_date	Should be a date value	Syntactic (data type)

Table 5 Test Cases to Assess Electronic Health Records

	Query
1	INSERT INTO tbl_test_results (test_id, status, description) values (SELECT 1 AS test_id, 'error' AS status, 'weight is negative' AS description FROM tbl_patients WHERE weight<0)
2	INSERT INTO tbl_test_results (test_id, status, description) values (SELECT 2 AS test_id, 'error' AS status, 'weight is nonnumeric' AS description FROM tbl_patients WHERE weight.type<>DOUBLE OR weight.type<>INTEGER OR weight.type<>FLOAT)
3	INSERT INTO tbl_test_results (test_id, status, description) values (SELECT 3 AS test_id, 'error' AS status, 'Sex is invalid' AS description FROM tbl_patients WHERE !(Sex='Male' OR Sex='Female' OR Sex='Other'))
4	INSERT INTO tbl_test_results (test_id, status, description) values (SELECT 4 AS test_id, 'error' AS status, 'Sex is null' AS description FROM tbl_patients WHERE Sex=null)
5	INSERT INTO tbl_test_results (test_id, status, description) values (SELECT 5 AS test_id, 'error' AS status, 'start date is greater than end date' AS description FROM tbl_patients WHERE Start_date>End_date)
6	INSERT INTO tbl_test_results (test_id, status, description) values (SELECT 6 AS test_id, 'error' AS status, 'Invalid dates' AS description FROM tbl_patients WHERE Start_date.type<>Date OR End_date.type<>Date)

Data profiling [7] and data auditing [36] are statistical analysis tools that verify the data quality properties to assess the data and detect business rule violations, as well as anomalies and contradictions in the data. These tools are often used for testing the quality of data at the sources with the goal of rectifying data before it is loaded to the target data warehouse [33].

There exist data validation tools that perform data quality tests focusing on the target data. Data warehouse projects are typically designed for specific business domains and it is difficult to define a generalized data quality assurance model applicable to all data warehouse systems. As a result, the existing data quality testing tools are developed either for a specific domain or for applying basic data quality checks that are applicable to all domains. Other generalized tools let users define their desired data quality rules.

Achilles [46] proposed by the OHDSI community [47] is an example that generates specific data quality tests for the electronic health domain. This tool defines 172 data quality rules and verifies them using queries as test cases. The tool checks the data in health data warehouses to ensure consistency with the OMOP common data model. It also uses rules that check the semantics of health data to be plausible based on its rule set. Table 5 shows some examples.

Loshin [48] provided a data validation engine called GuardianIQ that does not define specific data quality rules but allows users to define and manage their own expectations as business rules for data quality at a high level in an editor. As a result, this tool can be used in any data warehousing project. The tool transforms declarative data quality rules into queries that measures data quality conformance with their expectations. Each data is tested against the query set and scored across multiple dimensions. The scores are used for the measurement of levels of data quality, which calculates to what extent the data matches the user's expectations.

Informatica Powercenter Data Validation [49] is another example of a tool that generates data quality tests and is generalized for use in any data warehouse project. It allows users to develop their business rules rapidly without having any knowledge of programming. The test cases, which are a set of queries, are generated from the user's business rules to be executed against the data warehouse under test.

Gao et al. [11] compare the existing data quality validation tools for general use in terms of the operation environment, supported DBMSs or products, data validation checks, and case studies. All the tools discussed in Gao et al.'s paper provide basic data quality validations, such as null value, data constraint, and data type checks. However, they do not assure the completeness of their data quality checks through well-defined test adequacy criteria. In software testing, a test adequacy criterion is a predicate that determines what properties of a software application must be exercised to constitute a complete test. We can define the test adequacy criteria for data quality tests as the number of columns, tables or constraints exercised by the quality tests. The set of test cases (queries) must contain tests to verify the properties of all the columns in all the tables of the sources or the target data warehouse.

Furthermore, the fault finding ability of the data quality tests is not evaluated in any of the surveyed approaches. We suggest that new research approaches be developed using mutation analysis techniques [50] to evaluate the ability of data quality tests to detect possible faults in the data. In these techniques, a number of faults are injected into the data to see how many of the faults are detected by the tests. Table 6 shows a number of sample faults to inject into the data to violate the data quality properties we defined in this section.

Table 6 Sample Faults Injected into Health Data for Mutation Analysis

Property	**Fault Type**
Data type	Store a string value in a numeric field
Data constraint	Copy a record to have duplicate values for a primary key field
Data plausibility	Store a negative value in a weight field
Logical constraint	Set a pregnancy status to *true* for a male patient

4.1.2 Security Testing of Underlying Data

Security testing of underlying data is the process of revealing possible flaws in the security mechanisms that protect the data in a data storage area. The security mechanisms must be built into the data warehousing systems. Otherwise, if access control is only built into the front-end applications but not into the data warehouse, a user may bypass access control by directly using SQL queries or reporting tools on the data warehouse [51].

Every source database may have its access privileges defined for its data based on organizational requirements. Data loaded to the target data warehouse is supposed to maintain the same security for the corresponding data in the sources, while enforcing additional policies based on the data warehouse requirements. For example, if the personal information of the patients in a hospital is protected via specific techniques such as by defining user profiles or database access control [8], the same protection must be applied for the patient data transformed to the target health data warehouse. Additional access polices may be defined on the target health data warehouse to authenticate medical researchers who want to analyze the patient data.

Security testing of the underlying data in a data warehouse involves a comparison of the access privileges defined for the target data with the ones defined for the corresponding source data to determine whether all the required protections are correctly observed in the data warehouse. For this purpose, we can define security tests by formulating queries that return defined permissions associated with the data in both the sources and the target data warehouse, and compare the permissions for equivalent data using either manual or automatic techniques.

4.2 Testing the Data Model

As the data model is the foundation for any database, it is critical to get the model right because a flawed model directly affects the quality of information. Data model tests ensure that the design of the model follows its standards both conceptually and logically and meets the organizational specifications. Documentation for the source and target model help equip testers with the required information for the systematic testing of data models.

4.2.1 Functional Evaluation of the Data Model

In this evaluation activity, the quality of the data model design is verified to be consistent with organizational requirements of the sources or the data warehouse. Some of the approaches are general enough to assess any data model (relational, nonrelational, or dimensional), while there exist other approaches that evaluate a specific data model.

Hoberman [12] created a data model scorecard to determine the quality of any data model design that can be applied to both the source area and the target data warehouse. The scorecard is an inspection checklist that includes a number of questions and the score for each question. The number in front of each question represents the score of the question assigned by Hoberman. The organization places a value between 0 and the corresponding score on each question to determine to what extent the model meets the functional requirements. The following is a description of each question related to the functional evaluation of data models and the corresponding scores:

1. Does the model capture the requirements (15)? This ensures that the data model represents the organizational requirements.
2. Is the model complete (15)? This ensures that both the data model and its metadata (data model descriptive information) are complete with respect to the requirements.
3. Does the model match its schema (10)? This ensures that the detail (conceptual, logical, or physical) and the perspective (relational, dimensional, or NoSQL) of the model matches its definition.
4. Is the model structurally correct (15)? This validates the design practices (such as primary key constraints) employed for building the data model.
5. Are the definitions appropriate (10)? This ensures that the definitions in the data model are correct, clear, and complete.
6. Is the model consistent with the enterprise (5)? This ensures that the set of terminology and rules in data model context can be comprehended by the organization.
7. Does the metadata match the data (10)? This ensures that the data model's description is consistent with the data model.

Golfarelli and Rizzi [8] proposed three types of tests on the conceptual and logical dimensional data model in a data warehouse:

- A *fact test* verifies whether or not the conceptual schema meets the preliminary workload requirements. The preliminary workload is a set of queries that business users intend to run against the target data warehouse. These queries help the data warehouse designers identify required facts, dimensions, and measurements in the dimensional data model [52]. For each workload, the fact test checks whether or not the required measures are included in the fact schema. This evaluation also measures the number of nonsupported workloads.
- A *conformity test* assesses how well the conformed dimensions are designed in a dimensional data model. Such a model includes fact tables that keep metrics of a business process, and dimension tables that contain descriptive attributes. A fact table contains the keys to the dimension tables. A conformed dimension is one that relates to more than one fact. These dimensions support the ability to integrate data from multiple business processes. The conformity test is carried out by measuring the sparseness of a bus matrix [53] that is a high-level abstraction of a dimensional data model. In this matrix, columns are the dimension tables, and rows are the fact tables (business processes). The matrix associates each fact with its dimensions. If there is a column in the matrix with more than one nonzero element, it shows the existence of a conformed dimension. If the bus matrix is dense (i.e., most of the elements are nonzero), it shows that there are dimensions that are associated with many facts, which indicates that the model includes overly generalized columns. For example, a *person* column refers to a wide variety of people, from employees to suppliers and customers while there is zero overlap between these populations. In this case, it is preferable to have a separate dimension for each population and associate them to the corresponding fact. On the other hand, if the bus matrix is sparse (i.e., most of the elements are zero), it shows that there is a few conformed dimension defined in the dimensional model, which indicates that the model includes overly detail columns. For example, each individual descriptive attribute is listed as a separate column. In this case, it is preferable to create a conformed dimension that is shared by multiple facts.
- A *star test* verifies whether or not a sample set of queries in the preliminary workload can be correctly formulated in SQL using the logical data model. The evaluation measures the number of nonsupported workloads.

The above functional evaluation activities are manually performed via inspections. There is a lack of automated techniques.

4.2.2 Structural Evaluation of the Data Model

This type of testing ensures that the data model is correctly implemented using the database schema. The database schema is assessed for possible flaws. MySQL schema validation plug-in performs general validation for relational data models [54]. It evaluates the internal structure of the database schema and performs the following checks:

1. Validate whether content that is not supposed to be empty is actually empty. The tool reports an error if any of the following empty content exists in the relational database schema:
   - A table without columns
   - A view without SQL code
   - A table/view not being referenced by at least one role
   - A user without privileges
   - A table/object that does not appear in any ER diagrams
2. Validate whether a table is correctly defined by checking the primary key and foreign key constraints in that table. The tool reports an error if any of the following incorrect definition exists in the relational database schema:
   - A table without primary key
   - A foreign key with a reference to a column with a different type
3. Validate whether there are duplications in the relational database objects. The tool reports an error if any of the following duplications exist in the relational database schema:
   - Duplications in object names
   - Duplications in roles or user names
   - Duplications in indexes
4. Validate whether there are inconsistencies in the column names and their types. The tool reports an error if the following inconsistency exists in the relational database schema:
   - Using the same column name for columns of different data types

The above approach targets the structural validation of the relational data schema but it does not apply to nonrelational and other data schema.

To assess the coverage of validation, we suggest using various structural metrics. These metrics are predicates that determine what properties of a schema must be exercised to constitute a thorough evaluation. The metrics are the number of views, routines, tables, columns, and structural constraints that are validated during the structural evaluations.

4.2.3 Usability Evaluation of the Data Model

Usability evaluation of a data model tests whether the data model is easy to read, understand, and use by the database and data warehouse designers. A data model is usually designed in a way to cover the requirements of database and data warehouse designers. There are many common data models designed for specific domains, such as health, banking, or business. The Hoberman scorecard [12] discussed in the functional evaluation of the data model also includes a number of questions and their scores for usability evaluation of any data model. The data warehouse designer places a value between 0 and the corresponding score on each question to determine to what extent the model meets the usability requirements. The following is a description of each question related to the usability evaluation of data models and the corresponding scores:

1. Does the model use generic structures, such as data element, entity, and relationship (10)? This ensures that the data model uses appropriate abstractions to be transferable to more generic domains. For example, instead of using phone number, fax number, or mobile number elements, an abstract structure contains phone and phone type which accommodates all situations.
2. Does the model meet naming standards (5)? This ensures that the terms and naming conventions used in the model follow the naming standards for data models. For example, inconsistent use of uppercase letter, lowercase letter, and underscore, such as in Last Name, FIRST NAME, and middle_name, indicates that naming standards are not being followed.
3. Is the model readable (5)? This ensures that the data model is easy to read and understand. For example, it is more readable to group the data elements that are conceptually related into one structure instead of scattering the elements over unrelated structures. For example, city, state, and postal code are grouped together.

The above approach involves human inspection, and there does not exist automated techniques for the usability testing of relational, nonrelational, and dimensional data models.

4.2.4 Maintainability Evaluation of the Data Model

Due to the evolving nature of data warehouse systems, it is important to use a data model design that can be improved during the data warehouse lifecycle. Maintainability assessments evaluate the quality of a source or target data model with respect to its ability to support changes during an evolution process [55].

Calero et al. [56] listed metrics for measuring the complexity of a data warehouse star design that can be used to determine the level of effort required to maintain it. The defined complexity metrics are for the table, star, and schema levels. The higher the values, the more complex is the design of the star model, and the harder it is to maintain the model. The metrics are as follows:

- *Table metrics*
  - Number of attributes of a table
  - Number of foreign keys of a table
- *Star metrics*
  - Number of dimension tables of a star schema
  - Number of tables of a star schema that correspond to the number of dimension tables added to the fact table
  - Number of attributes of dimension tables of a star schema
  - Number of attributes plus the number of foreign keys of a fact table
- *Schema metrics*
  - Number of fact tables of the star schema
  - Number of dimension tables of the star schema
  - Number of shared dimension tables that is the number of dimension tables shared for more than one star of the schema
  - Number of the fact tables plus the number of dimension tables of the star schema
  - Number of attributes of fact tables of the star schema
  - Number of attributes of dimension tables of the star schema

These metrics give an insight into the design complexity of the star data model, but there is no information in the Calero et al. paper on how to relate maintainability tests to these metrics. There is also a lack of work in developing metrics for other data models such as the snowflake model or relational data models.

4.3 Testing Data Management Product

Using the right product for data management is critical to the success of data warehouse systems. There are many categories of products used in data warehousing, such as DBMSs, big data management systems, data warehouse appliances, and cloud data warehouses that should be tested to ensure that it is the right technology for the organization. In the following sections, we describe the existing approaches for performance, stress, and recovery testing of the data management products.

4.3.1 Performance and Stress Testing of Data Management Product

Performance testing determines how a product performs in terms of responsiveness under a typical workload [57]. The performance of a product is typically measured in terms of response time. This testing activity evaluates whether or not a product meets the efficiency specifications claimed by the organizations.

Stress testing evaluates the responsiveness of a data management product using an extraordinarily large volume of data by measuring the response time of the product. The goal is to assess whether or not the product performs without failures when dealing with a database with a size significantly larger than expected [8].

Due to the fact that the demand for real-time data warehouses [3] and real-time analysis is increasing, performance and stress testing play a major role in data warehousing systems. Due to the growing nature of data warehousing systems, the data management product tolerance must be evaluated using unexpectedly large volumes of data. The product tolerance is the maximum volume of data the product can manage without failures and crashes. Comparing efficiency and tolerance characteristics of several data management products help data warehouse designers choose the appropriate technology for their requirements.

Performance tests are carried out on both real data or mock (fake) datasets with a size comparable with the average expected data volume [8]. However, stress tests are carried out on mock databases with a size significantly larger than the expected data volume. These testing activities are performed by applying different types of requests on the real or mock datasets. A number of queries are executed, and the responsiveness of the data management product is measured using standard database metrics. An important metric is the maximum query response time because query execution plays an important role in data warehouse performance measures. Both simple and multiple join queries are executed to validate the performance of queries on databases with different data volumes. Business users develop sample queries for performance testing with specified acceptable response times for each query [1].

Slutz [58] developed an automatic tool called Random Generation of SQL (RAGS) that stochastically generates a large number of SQL Data Manipulation Language (DML) queries that can be used to measure how efficiently a data management system responds to those queries. RAGS generates the SQL queries by parsing a stochastic tree and printing the query out. The parser stochastically generates the tree as it traverses the tree using database information (table names, column names, and column types). RAGS generates 833 SQL queries per second that are useful for performance and stress testing purposes.

Most performance and stress testing approaches in the literature focus on DBMSs [59], but there is a lack of work in performance testing of data warehouse appliances or cloud data warehouses.

4.3.2 Recovery Testing of Data Management Product

This testing activity verifies the degree to which a data management product recovers after critical events, such as power disconnection during an update, network fault, or hard disk failures [8].

As data management products are the key components of any data warehouse systems, they need to recover from abnormal terminations to ensure that they present correct data and that there are no data loss or duplications.

Gunawi et al. [60] proposed a testing framework to test the recovery of cloud-based data storage systems. The framework systematically pushes cloud storage systems into 40,000 unique failures instead of randomly pushing systems into multiple failures. They also extended the framework to evaluate the expected recovery behavior of cloud storage systems. They developed a logic language to help developers precisely specify recovery behavior.

Most data warehousing systems that use DBMSs or other transaction systems rely on the atomicity, consistency, isolation, and durability (ACID) properties [61] of database transactions to meet reliability requirements. Database transactions allow correct recovery from failures and keep a database consistent even after abnormal termination. Smith and Klingman [62] proposed a method for recovery testing of transaction systems that use ACID properties. Their method implements a recovery scenario to test the recovery of databases affected by the scenario. The scenario uses a two-phase transaction process that includes a number of service requests and is initiated by a client application. The scenario returns to the client application without completing the processing of transaction and verifies whether or not the database has correctly recovered. The database status is compared to the expected status identified by the scenario.

4.4 Summary

Table 7 summarizes the testing approaches that have been applied to the sources and the target data warehouse that we discussed in this section.

Table 7 Testing the Sources and the Target Data Warehouse

Test Category	Component	GuardianIQ [48]	Informatica [49]	Hoberman [12]	Golfarelli and Rizzi [8]	MySQL plug-in [54]	Calero et. al, [56]	Slutz [58]	Gunawi et al. [60]	Smith and Klingman [62]
Functional	Underlying data Data model Product	✓	✓	✓	✓					
Structural	Data model					✓				
Usability	Data model			✓						
Maintainability	Data model							✓		
Performance	Product							✓		
Stress	Product							✓		
Recovery	Product								✓	
Recovery	Product									✓
Security	Underlying data									

Gray shaded rows indicate that we did not find approaches or tools to support that kind of testing activity even though they are necessary for a real-world data-warehouse.

There are no methods proposed for the security testing of underlying data in data warehouse systems (as indicated by the gray shaded row in the table).

We have identified the following open problems in testing the sources and the target data warehouse.

- In the area of *functional testing of underlying data*, there is no systematic way to assure the completeness of the test cases written/generated by different data quality assurance tools. We suggest that new research approaches be developed using a test adequacy criterion, such as number of fields, tables, or constraints as properties that must be exercised to constitute a thorough test.
- Data quality rules are not formally specified in the business requirements for *the functional testing of the underlying data*. A tester needs to bridge the gap between informal specifications and formal quality rules.
- It is difficult to design a generalized *data quality test* applicable to all data warehouse systems because data warehouse projects are typically designed for specific business domains. There are a number of generalized tools that let users define their desired data quality rules.
- The fault finding ability of the *data quality tests* are not evaluated in the literature. One can use mutation analysis techniques to perform this evaluation.
- No approach has been proposed for *the security testing of underlying data*. One can compare the access privileges defined for the target data with the ones defined for the corresponding source data to ensure that all the required protections are correctly observed in the target data warehouse.
- There is a lack of automatic *functional evaluation* techniques for data models. The existing functional evaluation activities are manually performed through human inspections.
- There is a lack of *structural evaluation* techniques for nonrelational and dimensional schema. The existing approaches focus on the relational data schema.
- No formal technique has been proposed for *the usability testing of data models*. The proposed approaches are typically human inspections.
- In the area of *maintainability testing of data models*, a number of design complexity metrics have been proposed to get an insight into the capability of the data model to sustain changes. However, there is no information on how to design maintainability tests based on the metrics.
- The heterogeneous data involved in the data warehousing systems make *the performance and stress testing of data management products* difficult. Testers

must use large datasets in order to perform performance and stress tests. Generating this voluminous data that reflect the real characteristics of the data is an open problem in these testing activities.

- There is a lack of work in *performance and stress testing of data warehouse appliance and cloud data warehouses*. The proposed approaches in the literature typically focus on testing DBMSs.

5. TESTING ETL PROCESS

This testing activity verifies whether or not the ETL process extracts data from sources, transforms it into an appropriate form, and loads it to a target data warehouse in a correct and efficient way. As the ETL process directly affects the quality of data transformed to a data warehouse [9], it has been the main focus of most data warehouse testing techniques [3]. In this section, we describe existing functional, performance, scalability, reliability, regression, and usability testing approaches as well as propose a new approach based on our experience in testing the ETL process in a health data warehouse [63].

5.1 Functional Testing of ETL Process

Functional testing of ETL process ensures that any changes in the source systems are captured correctly and propagated completely into the target data warehouse [3]. Two types of testing have been used for evaluating the functionality of ETL process, namely, data quality and balancing tests.

5.1.1 Data Quality Tests

This testing activity verifies whether or not the data loaded into a data warehouse through the ETL process is consistent with the target data model and the organizational requirements. Data quality testing focuses on the quality assessment of the data stored in a target data warehouse. Data quality tests are defined based on a set of quality rules provided by domain experts. These rules are based on both domain and target data model specifications to validate the syntax and semantics of data stored in a data warehouse. For example, in our health data warehouse project, we use data quality rules from six clinical research networks, such as Achilles [46] and PEDSnet [64] to write test cases as queries to test the data quality. Achilles and PEDSnet define a number of rules to assess the quality of electronic health records, and report errors and warnings based on the data. These quality rules are defined and periodically updated in a manner to fit the use and

Table 8 Examples of Achilles Data Quality Rules

Rule_id	Data Quality Rule	Status	Description
19	Year of birth should not be prior to 1800	Warning	Checks whether or not year of birth is less than 1800
32	Percentage of patients with no visits should not exceed a threshold value	Notification	Checks whether or not the percentage of patients that have no visit records is greater than 5

need of the health data users. Achilles defines its data quality rules as SQL queries while PEDSnet uses R. Table 8 shows examples of two data quality rules that are validated in Achilles.

Note that the tools described in Section 4.1.1 to test the quality of the underlying data in a data warehousing system can also be used to execute data quality tests for the ETL process. The difference is that in the context of ETL testing, the tools have a different purpose, which is to test any time data is added or modified through the ETL process.

5.1.2 Balancing Tests

Balancing tests ensure that the data obtained from the source databases is not lost or incorrectly modified by the ETL process. In this testing activity, data in the source and target data warehouse are analyzed and differences are reported.

The balancing approach called *Sampling* [65] uses source-to-target mapping documents to extract data from both the source and target tables and store them in two spreadsheets. Then it uses the *Stare and Compare* technique to manually verify data and determine differences through viewing or *eyeballing* the data. Since this task can involve the comparison of billions of records, most of the time, a few number of the entire set of records are verified through this approach.

IBM QuerySurge [65] is a commercial tool that was built specifically to automate the balancing tests through query wizards. The tool implements a method for fast comparison of validation query results written by testers [66]. The query wizards implement an interface to make sure that minimal effort and no programming skills are required for developing balancing tests and obtaining results. The tool compares data based on column, table, and record count properties. Testers select the tables and columns to be compared in the wizard. The problem with this tool is that it only compares data that is not modified during the ETL transformation, which is claimed to be 80% of data. However, the goal of ETL testing should also be to validate data that has been reformatted and modified through the ETL process.

Another method is *Minus Queries* [65] in which the difference between the source and target is determined by subtracting the target data from the source data to show existence of unbalanced data. The problem with this method is the potential for false positives. For example, as many data warehouses keep historical data, there may be duplicate records in the target data warehouse corresponding to the same entity and the result might report an error based on the differences in number of records in the source and the target data warehouse. However, these duplications are actually allowed in the target data warehouse.

We propose to identify discrepancies that may arise between the source and the target data due to an incorrect transformation process. Based on these discrepancies we define a set of properties, namely, completeness, consistency, and syntactic validity.

Completeness ensures that all the relevant source records get transformed to the target records. Consistency and syntactic validity ensure correctness of the transformation of the attributes. Consistency ensures that the semantics of the various attributes are preserved in the transformation process. Syntactic validity ensures that no problems occur due to the differences in the syntax between the source and the target data.

In our project, we generated balancing tests to compare the data in the health data warehouse, which uses a dimensional database on Google BigQuery, with the corresponding data in sources, which use dimensional patient databases of two hospitals [63].

The source-to-target mappings available in the ETL transformation specifications provide the necessary information to identify corresponding tables and attributes in the sources and target data warehouse and assist in developing an appropriate testing strategy [7]. The ETL transformations include one-to-one, many-to-one, and many-to-many mappings. We used the mapping documents from the health data warehouse to extract corresponding source and target tables and attributes, both for modified and nonmodified data. Then we generated a set of test assertions as queries to compare the source and target data verifying the proposed properties.

The fault finding ability of the balancing tests are not evaluated in any of the surveyed approaches. We proposed to use the mutation analysis technique [50] to evaluate the ability of our balancing tests in detecting possible

Table 9 Mutation Operators Used to Inject Faults in Health Data

Operator	Description
AR	Add random record
DR	Delete random record
MNF	Modify numeric field
MSF	Modify string field
MNF_{min}	Modify min of numeric field
MNF_{max}	Modify max of numeric field
MSF_{length}	Modify string field length
MF_{null}	Modify field to null

faults in the data. Table 9 shows mutation operators we proposed to inject faults into the data to assess the effectiveness of the balancing tests.

Due to the voluminous data involved in data warehousing, a comprehensive functional test of ETL must consider all possible inputs for an adequate testing. However, there is a limitation in defining adequate test inputs in the literature. As it is impossible to test the functionality of the ETL process with all possible test inputs, one can use systematic input space partitioning [67] techniques to generate test data. Input space partitioning is a software testing technique that groups the input data into partitions of equivalent data called *equivalent classes*. Test inputs can be derived from each partition. A comprehensive test must generate at least one input for each partition.

Moreover, ETL testers typically do not have access to real data because of the confidentiality of data in the sources, and they need to generate mock data that correctly represents the characteristics of the real data. There are a number of mock data generator tools, such as Mockaroo [68] and Databene Benerator [69] that randomly generate test data. However, testers must generate data in a systematic manner, such as through input space partitioning techniques to cover data from all equivalent classes.

5.2 Performance, Stress, and Scalability Testing of ETL Process

Performance tests assess whether or not the entire ETL process is performed within the agreed time frames by the organizations [70]. The goal of performance testing of ETL is to assess ETL processing time under typical workloads that are comparable with the average expected data volume [42].

Stress tests also assess the ETL processing time but under a workload which is significantly larger than the expected data volume. The goal of stress testing of ETL is to assess ETL tolerance by verifying whether or not it crashes or fails when dealing with an extraordinarily large volume of data.

Scalability testing of ETL is performed to assess the process in terms of its capability to sustain further growth in data warehouse workload and organizational requirements [1, 70]. The goal of scalability testing is to ensure that the ETL process meets future needs of the organization. Mathen [1] stated that this growth mostly includes an increase in the volume of data to be processed through the ETL. As the data warehouse workload grows, the organizations expect ETL to sustain extract, transform, and load times. Mathen [1] proposed an approach to test the scalability of ETL by executing ETL loads with different volumes of data and comparing the times used to complete those loads.

In all of these testing activities, the processing time of ETL is evaluated when a specific amount of data is extracted, transformed, and loaded into the data warehouse [8]. The goal is to determine any potential weaknesses in ETL design and implementation, such as reading some files multiple times or using unnecessary intermediate files or storage [1]. The initial extract and load process, along with the incremental update process must be evaluated through these testing activities.

Wyatt et al. [71] introduced two primary ways to measure the performance of ETL, i.e., time-based and workload-based. In the time-based method, they check if the ETL process was completed in a specific time frame. In the workload-based approach they test the ETL process using a known size of data as test data, and measure the time to execute the workload. Higher performing ETL processes will transfer the same volume of data faster. The two approaches can be blended to check if the ETL process was completed in a specific time frame using a specific size of data.

The above testing approaches test the entire ETL process under different workload conditions. However, the tests should also focus on the extraction, transformation, and load components separately and validate each component under specific workloads. For example, the performance testing of the extraction component verifies whether or not a typical sized data can be extracted from the sources in an expected time frame. Applying the tests separately on the constituent ETL components and procedures helps localize existing issues in the ETL design and implementation, and determine the areas of weaknesses. The weaknesses can be addressed by using an alternative technology, language, algorithm, or intermediate files. For example, consider the performance issue in the *Load* component of ETL, which incorrectly uses the full mode and loads the entire data every time instead of loading only the new added or modified data. In this case, the execution time of the *Load* component is considerably longer than the *Extract* and *Transform* components. This problem can be localized if we apply the tests on the individual components instead of on the entire ETL process.

5.3 Reliability Testing of ETL Process

This type of testing ensures the correctness of the ETL process under both normal and failure conditions [71]. Normal conditions represent situations in which there are no external disturbances or unexpected terminations in the ETL process. To validate the reliability of the ETL process under normal conditions, we want to make sure that given the same set of inputs and initial states, two runs of ETL will produce the same results. We can compare the two result sets using properties such as completeness, consistency, and validity.

Failure conditions represent abnormal termination of ETL as a result of loss of connection to a database or network, power failure, or a user terminating the ETL process. In such cases the process should be able to either complete the task later or restore the process to its starting point. To test the reliability of the ETL process under abnormal conditions, we can simulate the failure conditions and compare the results from a failure run with the results of a successful run to check if the results are correct and complete. For example, we should check that no records were loaded twice to the target data warehouse as a result of the failure condition followed by rerunning the ETL process.

Most ETL implementations indirectly demonstrate reliability features by relying on the ACID properties of DBMSs [71]. If DBMSs are used in a data warehousing system to implement the sources or the target data warehouse components, these components recover from problems in the Extract, Transform, or Load processes. However, there are data management products other than DBMSs used in the data warehousing systems that do not support ACID properties (e.g., Google Cloud Bigtable [72] that is a NoSQL data management product). In such cases, reliability needs to be addressed separately and appropriate test cases must be designed.

Note that balancing tests may be performed to compare the two result datasets in both normal and abnormal conditions to verify the proposed properties in addition to other reliability tests.

5.4 Regression Testing of ETL Process

Regression tests check if the system still functions correctly after a modification. This testing phase is important for ETL because of its evolving nature [8]. With every new data warehouse release, the ETL process needs to evolve to enable the extraction of data from new sources for the new applications. The goal of regression testing of ETL is to ensure that the enhancements and modifications to the ETL modules do not introduce new faults [73]. If a new program is added to the ETL, interactions between the new and old programs should be tested.

Manjunath et al. [74] automated regression testing of ETL to save effort and resources with a reduction of 84% in regression test time. They used Informatica [49] to automatically generate test cases in SQL format, execute test cases, and compare the results of the source and target data. However, their approach uses the *retest all* [75] strategy, which reruns the entire set of test cases for regression testing of the ETL process. Instead, they could use *regression test selection* [75] techniques to run a subset of test cases to test only the parts of ETL that are affected by project changes. These techniques classify the set of test cases into retestable and reusable tests for regression testing purposes in order to save testing cost and time. A retestable test case tests the modified parts of the ETL process and needs to be rerun for the safety of regression testing. A reusable test case tests the unmodified parts of the ETL process and does not need to be rerun, while it is still valid [76].

Mathen [1] proposed to perform regression testing by storing test inputs and their results as expected outputs from successful runs of ETL. One can use the same test inputs to compare the regression test results with the previous results instead of generating a new set of test inputs for every regression test [1].

5.5 Usability Testing of ETL Process

The ETL process consists of various components, modules, databases, and intermediate files with different technologies, DBMSs, and languages that require many prerequisites and settings to be executed on different platforms. ETL is not a one-time process; it needs to be executed frequently or any time data is added or modified in the sources. As a result, it is important to execute the entire process with configurations that are easy to set up and modify.

Usability testing of ETL process assesses whether or not the ETL process is easy to use by the data warehouse implementer. This testing activity determines how easy it is to configure and execute ETL in a data warehouse project.

We suggest to assess the usability of the ETL process by measuring the manual effort involved in configuring ETL in terms of time. The configuration effort includes (1) providing connection information to the sources, data staging area, or target data warehouse, (2) installing prerequisite packages, (3) preprocessing of data before starting the ETL process, and (4) human interference to execute jobs that run each of the extract, transform, and load components. We can also do a survey with different users that gives us more information about the difficulty level.

5.6 Summary

Table 10 summarizes the existing approaches to test different aspects of the ETL process. As can be seen from the table, scalability, reliability, and usability testing were not reported in the literature even though they are critical for a comprehensive testing of the process. We identified the following open problems in ETL testing. We summarize areas and ideas for future investigation.

- In *the functional testing of ETL*, there is not a systematic way to assure the completeness of the test cases written/generated as a set of queries. As with the functional testing of underlying data, we can use appropriate test adequacy criteria to evaluate and create a thorough test.
- There is a lack of systematic techniques to generate mock test inputs for *the functional testing of the ETL process*. We can use input space partitioning techniques to generate test data for all the equivalent classes of data. Current tools generate random test data with not much similarity with the characteristics of real data.
- The fault finding ability of *the balancing tests* is not evaluated in the surveyed approaches. We can use mutation analysis for this evaluation.
- In *the performance, stress, and scalability testing of ETL*, the existing approaches test the entire ETL process under different workloads. We can apply tests to the individual components of ETL to determine the areas of weaknesses.

Table 10 Testing Extract, Transform, and Load (ETL)

Testing Category	GuardianIQ [48]	Informatica [49]	QuerySurge [65]	Wyatt et al. [71]	Mathen [1]	Manjunath et al. [74]
Functional	✓	✓	✓			
Performance				✓		
Stress						
Scalability					✓	
Reliability						
Regression						✓
Usability						

Gray shaded rows indicate that we did not find approaches or tools to support that kind of testing activity even though they are necessary for a real-world data-warehouse.

- The heterogeneous data involved in the data warehousing systems make *the performance, stress, and scalability testing of the ETL process* difficult. Testers must use large heterogeneous datasets in order to perform tests.
- The existing ETL implementations rely on ACID properties of transaction systems, and ignore *the reliability testing of the ETL process*. We can perform the balancing tests proposed in Section 5.1.2 to compare the results of the ETL process under normal conditions with the ones under abnormal conditions to verify the properties, namely, completeness, consistency, and syntactic validity.
- No approach has been proposed to *test the usability of the ETL process*. We define this testing activity as the process of determining whether or not the ETL process is easy to use by the data warehouse implementer. One can test the usability of the ETL process by assessing the manual effort involved in configuring ETL that is measured in terms of time.

6. TESTING FRONT-END APPLICATIONS

Front-end applications in data warehousing are used by data analyzers and researchers to perform various types of analysis on data and generate reports. Thus, it is important to test these applications to make sure the data are correctly, effectively, and efficiently presented to the users.

6.1 Functional Testing of Front-End Applications

This testing activity ensures that the data is correctly selected and displayed by the applications to the end users. The goal of testing the functionality of the front-end applications is to recognize whether the analysis or end result in a report is incorrect, and whether the cause of the problem is the front-end application rather than the other components or processes in the data warehouse.

Golfarelli and Rizzi [8] compared the results of analyses queries displayed by the application with the results obtained by executing the same queries directly (i.e., without using the application as an interface) on the target data warehouse. They suggested two different ways to create test cases as queries for functionality testing. In a black-box approach, test cases are a set of queries based on user requirements. In a white-box approach, the test cases are determined by defining appropriate coverage criteria for the dimensional data. For example, test cases are created to test all the facts, dimensions, and attributes of the dimensional data.

The approaches proposed by Golfarelli and Rizzi are promising. In our project, we have used Achilles, which is a front-end application that performs quality assurance and analysis checks on health data warehouses in the OMOP [77] data model. The queries in Achilles are executable on OMOP.

The functional testing of the front-end applications must consider all possible test inputs for adequate testing. As it is impossible to test the functionality of the front-end applications with all possible test inputs, we can use systematic input space partitioning [67] techniques to generate test data.

6.2 Usability Testing of Front-End Applications

Two different aspects of usability of front-end applications need to be evaluated during usability testing, namely, ease of configuring and understandability.

First, we must ensure that the front-end application can be easily configured to be connected to the data warehouse. The technologies used in the front-end application should be compatible with the ones used in the data warehouse; otherwise, it will require several intermediate tools and configurators to use the data warehouse as the application's back-end. For example, if an application uses JDBC drivers to connect to a data warehouse, and the technology used to implement the data warehouse does not support JDBC drivers, it will be difficult to connect the front-end apps to the back-end data warehouse. We may need to reimplement parts of the application that set up connections to the data warehouse or change the query languages that are used. We suggest evaluating this usability characteristic by measuring the time and effort required to configure the front-end application and connect it to the target data warehouse.

Second, we must ensure that the front-end applications are understandable by the end users [70], and the reports are represented and described in a way that avoids ambiguities about the meaning of the data. Existing approaches to evaluate the usability of generic software systems [78] can be used to test this aspect of front-end applications. These evaluations involve a number of end users to verify the application. Several instruments are utilized to gather feedback from users on the application being tested, such as paper prototypes [79], and pretest and posttest questionnaires.

6.3 Performance and Stress Testing of Front-End Applications

Performance testing evaluates the response time of front-end applications under typical workloads, while stress testing evaluates whether the application performs without failures under significantly heavy workloads. The workloads are identified in terms of number of concurrent users, data volumes, and number of queries. The tests provide various types of workloads to the front-end applications to evaluate the application response time.

Filho et al. [80] introduced the OLAP Benchmark for Analysis Services (OBAS) that assesses the performance of OLAP analysis services responsible for the analytical process of queries. The benchmark uses a workload-based evaluation that processes a variable number of concurrent executions using variable-sized dimensional datasets. It uses the Multidimensional Expressions (MDX) [81] query language to perform queries over multidimensional data cubes.

Bai [82] presented a performance testing approach that assesses the performance of reporting systems built using the SQL Server Analysis Services (SSAS) technology. The tool uses the MDX query language to simulate user requests under various cube loads. Metrics such as average query response time and number of queries answered are defined to measure the performance of these types of reporting services.

The above two tools compare the performance of analysis services such as SSAS or Pentaho Mondrian. However, the tools do not compare analysis services that support communication interfaces other than XML for Analysis (XMLA), such as OLAP4J.

6.4 Summary

Table 11 summarizes existing approaches that test front-end applications in data warehousing systems. Although it is important to assess usability, none of the approaches addressed usability testing. Stress testing of the front-end applications is not reported in the surveyed approaches. Below

Table 11 Testing Front-End Applications

Test Category	Golfarelli and Rizzi [8]	Filho et al. [80]	Bai [82]
Functional	✓		
Usability			
Performance		✓	✓
Stress			

Gray shaded rows indicate that we did not find approaches or tools to support that kind of testing activity even though they are necessary for a real-world data-warehouse.

is a summary of the open problems in testing front-end applications, and ideas for future investigation.

- Existing approaches proposed for *functionality testing of the front-end applications* compare the results of queries in the application with the ones obtained by directly executing the same queries on the target data warehouse.
- *Functional testing of the front-end applications* must consider all types of test inputs. We can use input space partitioning techniques to generate test data for this testing activity.
- To support *usability testing of front-end applications*, we define a new aspect of testing to assess how easy it is to configure the application. We can test this aspect of usability by measuring the manual effort involved in configuring the front-end application in terms of time.
- The voluminous data involved in the data warehousing systems makes *performance and stress testing of the front-end applications* difficult. Testers must use large datasets in order to perform realistic tests.

7. CONCLUSION

In this chapter, we described the challenges, approaches, and open problems in the area of testing data warehouse components. We described the components of a data warehouse using examples from a real-world health data warehouse project. We provided a classification framework that takes into account *what* component of a data warehouse was tested, and *how* the component was tested using various functional and nonfunctional testing and evaluation activities. We surveyed existing approaches to test and evaluate each component. Most of the approaches that we surveyed adapted traditional testing and evaluation approaches to the area of data warehouse testing. We identified gaps in the literature and proposed directions for further research. We observed that the following testing categories are open research areas.

- *Security testing* of the underlying data in the source and target components
- *Reliability testing* of the ETL process
- *Usability testing* of the ETL process
- *Usability testing* of the front-end applications
- *Stress testing* of the front-end applications

Future research needs to focus on filling the above gaps for comprehensively testing data warehouses. Moreover, the following techniques need to be developed or improved in all the testing activities in order to enhance the overall verification and validation of the data warehousing systems.

Test automation needs to improve to decrease the manual effort involved in data warehouse testing by providing effective test automation tools. The data involved in data warehouse testing are rapidly growing. This makes it impossible to efficiently test data warehouses while relying on manual activities. Existing testing approaches require a lot of human effort in writing test cases, executing tests, and reporting results. This makes it difficult to run tests repeatedly and consistently. Repeatability is a critical requirement of data warehouse testing because we need to execute the tests whenever data are added or modified in a data warehouse. The approaches previously discussed in this chapter are based on statistical analysis, manual inspections, or semiautomated testing tools that still need manual effort for generating test input values and assertions. Existing approaches to software test automation can be utilized to fully automate the tasks involved in data warehouse testing. However, automatic test assertion generation (oracle problem) is an open problem for software systems in general [83] because the expected test outputs for all possible test inputs are typically not formally specified. Testers often manually identify the expected outputs using informal specifications or their knowledge of the problem domain. The same problem exists for automatically generating test assertions for testing the data warehouse components. If the expected outputs are not fully specified in the source-to-target transformation rules or in data warehouse documentation, it will be difficult to automatically generate test assertions. Future research needs to fill the gap between informal specifications and formally specified outputs to automatically generate test assertions.

Like other generic software systems, data warehouse projects need to implement agile development processes [84], which help produce results faster for end users and adapt the data warehouse to ever-changing user requirements [85]. The biggest challenge for testing an agile data warehouse project is that the data warehouse components are always changing. As a result, testing needs to adapt as part of the development process. The design and execution of these tests often take time that agile projects typically do not have. The correct use of regression test selection techniques [75] can considerably reduce the time and costs involved in the iterative testing of agile data warehousing systems. These techniques help reduce costs by selecting and executing only a subset of tests to verify the parts of the data warehouse that are affected by the changes. At the same time, testers need to take into account trade-offs between the cost of selecting and executing tests, and the fault detection ability of the executed tests [75].

There will be a growing demand for real-time analysis and data requests [3]. The data warehouse testing speed needs to increase. Most of the existing functional and nonfunctional testing activities rely on testing the entire source, target, or intermediate datasets. As the data are incrementally extracted, transformed, and loaded into the target data warehouse, tests need to be applied to only the newly added or modified data in order to increase the speed of testing. Testing with the entire data should be applied only in the initial step where the entire data are extracted from the sources, transformed, and loaded to the target data warehouse for the first time.

Most of the existing tools rely on using real data as test inputs, while testers typically do not have access to the real data because of privacy and confidentiality concerns. Systematic test input generation techniques for software systems can be used in future studies to generate mock data with the characteristic of the real data and with the goal of adequately testing the data warehouse components. For example, we proposed to use random mock data generation tools that populate a database with randomly generated data while obeying data types and data constraints. Genetic and other heuristic algorithms have been used in automatic test input generation for generic software systems [86] with the goal of maximizing test coverage. The same idea can be utilized in data warehouse testing to generate test data for testing different components with the goal of maximizing test coverage for the component under test.

Identifying a test adequacy criterion helps testers evaluate their set of tests and improve the tests to cover uncovered parts of the component under test. Determining adequate test coverage is a limitation of current testing approaches. Further research needs to define test adequacy criteria to assess the completeness of test cases written or generated for different testing purposes. For example, test adequacy criteria for testing the underlying data can be defined as the number of tables, columns, and constraints that are covered during a test activity. The adequacy criteria can also be defined as the number of data quality rules that are verified by the tests. Test adequacy criteria for white-box testing of the ETL process can be defined as the number of statements or branches of the ETL code that are executed during the ETL tests.

The fault finding abilities of existing testing approaches need to be evaluated. Mutation analysis techniques [50] can be used in future studies to evaluate the number of injected faults that are detected using the written/generated test cases. These techniques systematically seed a number of faults that simulate real faults in the program under test. The techniques execute the tests and determine whether or not the tests can detect the injected faults. Faults can be injected into both the code and the data in a data warehousing system. Different functional and nonfunctional tests are supposed to fail as a result of the injected faults. For example, balancing tests should result in failures because of imbalances caused by the seeded data faults. Functional testing of front-end application must fail due to the incorrect data that is reported in the final reports and analysis. A fault in the ETL code may result in the creation of unnecessary intermediate files during the ETL process and cause the performance tests to fail. Using these techniques help testers evaluate test cases and improve them to detect undetected faults.

Due to the widespread use of confidential data in data warehousing systems, security is a major concern. Security testing of all the components of data warehouses must play an important role in data warehouse testing. There are different potential security challenges in data warehousing systems that need to be addressed in future studies.

First, there are many technologies involved in the data warehousing implementations. Different DBMSs, data warehouse products, and cloud systems are being used to store and manage data of the sources, the intermediate DSA, and the target data warehouse. Correctly transforming data access control and user privileges from one technology to the other is a significant challenge in the security of the data warehousing systems. Future research in security testing needs to develop techniques that compare the privileges defined in the sources and the ones defined in the target of a transformation to ensure that all the privileges are correctly transformed without losing any information.

Second, due to the large number of interactive processes and distributed components involved in data warehousing systems, especially those containing sensitive data, there are many potential security attacks [87], such as man-in-the-middle, data modification, eavesdropping, or denial-of-service. The goal of such attacks may be to read confidential data, modify the data that results in misleading or incorrect information in the final reports, or disrupting any service provided by the data warehousing system. Some of the consequences can be detected using the previously discussed functional and nonfunctional testing approaches. For example, if the data were modified through an attack, balancing tests (Section 5.1.2) can detect the faulty data in the target data warehouse. However, comprehensive security techniques can prevent these types of attacks before the problem is propagated to the target data warehouse where error detection and fault localization is much more expensive. Security testing should detect vulnerabilities in the code, hardware, protocol implementations, and database access controls in a data warehousing project and report them to the data warehouse developers to avoid the exploitation of those vulnerabilities.

An alternative to data warehouse testing is to develop the data warehouse in a way that proves that the data warehouse implements the specifications and it will not fail under any circumstances. This approach is called *correct by construction* [88] in software engineering context. It guarantees the correct construction of software, and thus, does not require testing. Data warehousing systems include different distributed components, programs, and processes on various platforms that are implemented using different technologies. The large number of factors that affect data warehousing systems at run-time makes it practically impossible to prove that a data warehouse meets its specifications under all circumstances. Testing must be performed to validate the data warehousing systems in different situations. As a result, data warehouse testing is likely to be an active research field in the near future.

REFERENCES

[1] M.P. Mathen, Data warehouse testing, Infosys DeveloperIQ Magazine (2010) 1–8.
[2] N. Dedic, C. Stanier, An evaluation of the challenges of multilingualism in data warehouse development, in: 18th International Conference on Enterprise Information Systems, Rome, Italy, ISBN: 978-989-758-187-8, 2017, pp. 196–206.
[3] V. Rainardi, Building a Data Warehouse with Examples in SQL Server, first ed., Apress, 2008, ISBN: 1590599314, 9781590599310, pp. 477–489.
[4] M.J. Berry, G. Linoff, Data Mining Techniques: For Marketing, Sales, and Customer Support, second, John Wiley & Sons, Inc., 1997. ISBN: 978-0-471-17980-1
[5] A.V. Aho, J.D. Ullman, Foundations of Computer Science, C Edition, W. H. Freeman, 1994, ISBN: 978-0-7167-8284-1.
[6] J. Han, E. Haihong, G. Le, J. Du, Survey on NoSQL database, in: 6th International Conference on Pervasive Computing and Applications, 2011, pp. 363–366.
[7] D. Vucevic, W. Yaddow, Testing the Data Warehouse Practicum: Assuring Data Content, Data Structures and Quality, Trafford Publishing, 2012. ISBN: 978-1-4669-4356-8.
[8] M. Golfarelli, S. Rizzi, A comprehensive approach to data warehouse testing, in: 12th ACM International Workshop on Data Warehousing and OLAP, New York, NY, USA, ISBN: 978-1-60558-801-8, 2009, pp. 17–24.
[9] N. ElGamal, A. ElBastawissy, G. Galal-Edeen, Data warehouse testing, in: The Joint EDBT/ICDT Workshops, New York, USA, ISBN: 978-1-4503-1599-9, 2013, pp. 1–8.
[10] H.M. Sneed, Testing a datawarehouse—an industrial challenge, in: Academic Industrial Conference—Practice And Research Techniques, 2006, pp. 203–210.
[11] J. Gao, C. Xie, C. Tao, Big data validation and quality assurance—issuses, challenges, and needs, in: IEEE Symposium on Service-Oriented System Engineering, 2016, pp. 433–441.
[12] S. Hoberman, Data Model Scorecard: Applying the Industry Standard on Data Model Quality, first ed., Technics Publications, 2015, ISBN: 978-1-63462-082-6.
[13] Q. Li, Y.-L. Chen, Entity-relationship diagram, in: Modeling and Analysis of Enterprise and Information Systems, Springer, Berlin, Heidelberg, ISBN: 978-3-540-89555-8, 978-3-540-89556-5, 2009, pp. 125–139.
[14] M. Varga, On the differences of relational and dimensional data model, in: 12th International Conference on Information and Intelligent Systems, 2001, pp. 245–251.

[15] R. Kimball, M. Ross, W. Thornthwaite, J. Mundy, B. Becker, The Data Warehouse Lifecycle Toolkit, second ed., Wiley, 2008, ISBN: 978-0-470-14977-5.
[16] V. Gopalkrishnan, Q. Li, K. Karlapalem, Star/snow-flake schema driven object-relational data warehouse design and query processing strategies, in: DataWarehousing and Knowledge Discovery, Springer, Berlin, Heidelberg, 1999, pp. 11–22.
[17] Deleted in review.
[18] A. Askoolum, Structured query language, in: System Building With APL + WinJohn Wiley & Sons, Ltd, ISBN: 978-0-470-03434-7, 2006, pp. 447–477.
[19] F. Hinshaw, Data warehouse appliances: driving the business intelligence revolution, DM Review Magazine (2004) 30–34.
[20] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58. ISSN: 0001-0782.
[21] MySQL, https://www.mysql.com (accessed 02.05.17).
[22] Microsoft SQL Server, https://www.microsoft.com/sql-server (accessed 02.05.17).
[23] PostgreSQL, https://www.postgresql.org (accessed 02.05.17).
[24] Apache Accumulo, https://accumulo.apache.org (accessed 02.05.17).
[25] ArangoDB: Highly Available Multi-model NoSQL Database, https://www.arangodb.com (accessed 02.05.17).
[26] MongoDB, https://www.mongodb.com (accessed 02.05.17).
[27] Apache Hadoop, https://hadoop.apache.org (accessed 02.05.17).
[28] Oracle: Integrated Cloud Applications and Platform Services, https://www.oracle.com (accessed 02.05.17).
[29] IBM PureApplication, http://www-03.ibm.com/software/products/pureapplication (accessed 02.05.17).
[30] BigQuery: Analytics Data Warehouse, https://cloud.google.com/bigquery (accessed 02.05.17).
[31] Amazon Redshift | Data Warehouse Solution, https://aws.amazon.com/redshift (accessed 02.05.17).
[32] Observational Medical Outcomes Partnership, http://omop.org (accessed 14.04.17).
[33] R. Kimball, J. Caserta, The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data, first ed., Wiley, 2004. ISBN: 978-0-7645-6757-5.
[34] J. Barateiro, H. Galhardas, A survey of data quality tools, Datenbank Spektrum 14 (2005) 15–21.
[35] M.L. Lee, H. Lu, T.W. Ling, Y.T. Ko, Cleansing data for mining and warehousing, in: 10th International Conference on Database and Expert Systems Applications, Springer, Berlin, Heidelberg, 1999, pp. 751–760.
[36] S. Chaudhuri, U. Dayal, An overview of data warehousing and OLAP technology, ACM SIGMOD Record 26 (1) (1997) 65–74. ISSN: 0163-5808.
[37] G. Colliat, OLAP, relational, and multidimensional database systems, ACM SIGMOD Record 25 (3) (1996) 64–69, ISSN: 0163-5808.
[38] Crystal Reports: Formatting Multidimensional Reporting Against OLAP Data, http://www.informit.com/articles/article.aspx?p=1249227 (accessed 10.10.17).
[39] J. Iavindrasana, G. Cohen, A. Depeursinge, H. Muller, R. Meyer, A. Geissbuhler, Clinical data mining: a review, Yearb. Med. Inform. 48 (1) (2009) 121–133. ISSN: 0943-4747. https://imia.schattauer.de/contents/archive/issue/2347/issue/special/manuscript/11863/show.html.
[40] J.P. Shim, M. Warkentin, J.F. Courtney, D.J. Power, R. Sharda, C. Carlsson, Past, present, and future of decision support technology, Decis. Support. Syst. 33 (2) (2002) 111–126, ISSN: 0167-9236.

[41] E.S. Berner, Clinical Decision Support Systems: Theory and Practice, second ed., Springer, 2006, ISBN: 978-0-387-33914-6.
[42] M. Golfarelli, S. Rizzi, Data warehouse testing: a prototype-based methodology, Inf. Softw. Technol. 53 (11) (2011) 1183–1198, ISSN: 0950-5849.
[43] M.P. Neely, Data Quality Tools for Data Warehousing—A Small Sample Survey, State University of New York at Albany, 1998.
[44] M.A. Weiss, Data Structures & Algorithm Analysis in C++, fourth, Pearson, 2013. ISBN: 978-0-13-284737-7.
[45] M.G. Kahn, T.J. Callahan, J. Barnard, A.E. Bauck, J. Brown, B.N. Davidson, H. Estiri, C. Goerg, E. Holve, S.G. Johnson, S.-T. Liaw, M. Hamilton-Lopez, D. Meeker, T.C. Ong, P. Ryan, N. Shang, N.G. Weiskopf, C. Weng, M.N. Zozus, L. Schilling, A harmonized data quality assessment terminology and framework for the secondary use of electronic health record data, EGEMS (Washington, DC) 4 (1) (2016) 1244.
[46] OHDSI/Achilles, https://github.com/OHDSI/Achilles (accessed 11.05.17).
[47] G. Hripcsak, J.D. Duke, N.H. Shah, C.G. Reich, V. Huser, M.J. Schuemie, M.A. Suchard, R.W. Park, I.C.K. Wong, P.R. Rijnbeek, J. van der Lei, N. Pratt, G.N. Norén, Y.-C. Li, P.E. Stang, D. Madigan, P.B. Ryang, Observational Health Data Sciences and Informatics (OHDSI): opportunities for observational Researchers, Stud. Health Technol. Inform. 216 (2015) 574–578, ISSN: 0926-9630.
[48] D. Loshin, Rule-based data quality, in: 11th ACM International Conference on Information and Knowledge Management, New York, NY, USA, ISBN: 978-1-58113-492-6, 2002, pp. 614–616.
[49] Informatica, https://www.informatica.com/ (accessed 14.04.17).
[50] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE Trans. Softw. Eng. 37 (5) (2011) 649–678. ISSN: 0098-5589.
[51] K.B. Edwards, G. Lumpkin, Security and the Data Warehouse, Tech. rep., Oracle, 2005.
[52] M.K. Mohania, A.M. Tjoa, Data Warehousing and Knowledge Discovery: 12th International Conference, DaWaK, Springer, 2010, ISBN: 978-3-642-15104-0.
[53] M. Golfarelli, S. Rizzi, Data warehouse design: modern principles and methodologies, first, McGraw-Hill, Inc., 2009, ISBN: 978-0-07-161039-1
[54] MySQL: MySQL Workbench Manual: 9.2.3 Schema Validation Plugins, https://dev.mysql.com/doc/workbench/en/wb-validation-plugins.html (accessed 14.04.17).
[55] G. Papastefanatos, P. Vassiliadis, A. Simitsis, Y. Vassiliou, Design metrics for data warehouse evolution, in: International Conference on Conceptual Modeling, Springer, Berlin, Heidelberg, 2008, pp. 440–454.
[56] C. Calero, M. Piattini, C. Pascual, M.A. Serrano, Towards data warehouse quality metrics, in: The International Workshop on Design and Management of Data Warehouses Interlaken, Switzerland, 2001, pp. 1–10.
[57] B. Erinle, Performance testing with JMeter 2.9, first ed., Packt Publishing Ltd, 2013, ISBN: 978-1-78216-584-2, 978-1-78216-585-9.
[58] D.R. Slutz, Massive Stochastic Testing of SQL, in: 24th International Conference on Very Large Data Bases, San Francisco, CA, USA, ISBN: 978-1-55860-566-4, 1998, pp. 618–622.
[59] D. Chays, Y. Deng, P.G. Frankl, S. Dan, F.I. Vokolos, E.J. Weyuker, An agenda for testing relational database applications, Softw. Test. Verif. Rel. 14 (1) (2004) 17–44, ISSN: 0960-0833.
[60] H.S. Gunawi, T. Do, P. Joshi, P. Alvaro, J.M. Hellerstein, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, K. Sen, D. Borthakur, FATE and DESTINI: a framework for cloud recovery testing, in: 8th USENIX Conference on Networked Systems Design and Implementation, 2011, pp. 238–252.
[61] T. Haerder, A. Reuter, Principles of transaction-oriented database recovery, ACM Comput. Surv. 15 (4) (1983) 287–317, ISSN: 0360-0300.

[62] B. Smith, V.J. Klingman. Method and Apparatus for Testing Recovery Scenarios in Global Transaction Processing Systems, US Patent 707/999.202, 1997.
[63] H. Homayouni, An Approach for Testing the Extract-Transform-Load Process in Data Warehouse Systems (Master's thesis), Department of Computer Science, Colorado State University, 2018, Available at http://www.cs.colostate.edu/etl/papers/Thesis.pdf.
[64] PEDSnet, https://pedsnet.org/ (accessed 11.05.17).
[65] QuerySurge: Big Data Testing, ETL Testing & Data Warehouse Testing, http://www.querysurge.com/ (accessed 20.09.17).
[66] M. Marin, A data-agnostic approach to automatic testing of multi-dimensional databases, in: 7th International Conference on Software Testing, Verification and Validation 2014, pp. 133–142.
[67] I. Burnstein, Practical Software Testing: A Process-Oriented Approach, Springer, 2003, ISBN: 978-0-387-95131-7.
[68] Mockaroo - Random Data Generator|CSV/JSON/SQL/Excel, https://mockaroo.com/ (accessed 15.05.17).
[69] Databene Benerator, http://databene.org/databene-benerator/ (accessed 15.05.17).
[70] J. Theobald, Strategies for testing data warehouse applications, Inf. Manage. 17 (6) (2007) 20.
[71] L. Wyatt, B. Caufield, D. Pol, Principles for an ETL benchmark, in: Performance Evaluation and Benchmarking. Springer, Berlin, Heidelberg, 2009, pp. 183–198.
[72] Google Cloud Bigtable, https://cloud.google.com/bigtable/ (accessed 11.05.17).
[73] L.T. Moss, S. Atre, Business Intelligence Roadmap: The Complete Project Lifecycle for Decision-Support Applications, Addison-Wesley Professional, 2003. ISBN: 978-0-201-78420-6.
[74] T.N. Manjunath, R.S. Hegad, H.K. Yogish, R.A. Archana, I.M. Umesh, A case study on regression test automation for data warehouse quality assurance, Int. J. Inf. Technol. Knowl. Manage. 5 (2) (2012) 239–243.
[75] T.L. Graves, M.J. Harrold, J.-M. Kim, A. Porter, G. Rothermel, An empirical study of regression test selection techniques, ACM Trans. Softw. Eng. Methodol. 10 (2) (2001) 184–208. ISSN: 1049-331X.
[76] L.C. Briand, Y. Labiche, G. Soccar, Automating impact analysis and regression test selection based on UML designs, in: International Conference on Software Maintenance2002, pp. 252–261.
[77] Observational Medical Outcomes Partnership Common Data Model (OMOP CDM), https://www.ohdsi.org/data-standardization/the-common-data-model, (accessed 20.09.17).
[78] J.S. Dumas, J.C. Redish, A Practical Guide to Usability Testing, Rev Sub Edition, Intellect Ltd, 1999. ISBN: 978-1-84150-020-1.
[79] C. Snyder, Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces, first ed., Morgan Kaufmann, 1877.
[80] B.E.M. de Albuquerque Filho, T.L.L. Siqueira, V.C. Times, OBAS: an OLAP benchmark for analysis services., J. Inf. Data Manag. 4 (3) (2013) 390, ISSN: 21787107.
[81] B.-K. Park, H. Han, I.-Y. Song, XML-OLAP: a multidimensional analysis framework for XML warehouses, in: Data Warehousing and Knowledge DiscoverySpringer, Berlin, Heidelberg, 2005, pp. 32–42.
[82] X. Bai, Testing the performance of an SSAS cube using VSTS, in: 7th International Conference on Information Technology: New Generations, Las Vegas, NV, USA, 2010, pp. 986–991.
[83] E.T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo, The oracle problem in software testing: a survey, IEEE Trans. Softw. Eng. 41 (5) (2015) 507–525.
[84] D. Greer, Y. Hamon, Agile software development, Softw. Pract. Exp. 41 (9) (2011) 943–944, ISSN: 1097-024X.

[85] L. Corr, J. Stagnitto, Agile Data Warehouse Design: Collaborative Dimensional Modeling, From Whiteboard to Star Schema, third ed., DecisionOne Press, 2011, ISBN: 978-0-9568172-0-4.
[86] R.P. Pargas, M.J. Harrold, R.R. Peck, Test-data generation using genetic algorithms, Softw. Test. Verif. Rel. 9 (1999) 263–282.
[87] H.C.A. Van Tilborg, S. Jajodia, Encyclopedia of Cryptography and Security, second ed., Springer, 2011, ISBN: 978-1-4419-5905-8.
[88] L.P. Carloni, K.L. McMillan, A. Saldanha, A.L. Sangiovanni-Vincentelli, A methodology for correct-by-construction latency insensitive design, in: IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (Cat. No. 99CH37051)1999, pp. 309–315.

ABOUT THE AUTHORS

Hajar Homayouni is a Ph.D. student in Computer Science at Colorado State University. She received the Master's degree in Computer Science from Colorado State University in 2018. Her research interests are in software testing, data warehousing, Extract-Transform-Load design and testing, data quality testing, and machine learning.

Sudipto Ghosh is an Associate Professor in the Computer Science Department at Colorado State University. He received the Ph.D. degree in Computer Science from Purdue University, USA, in 2000. His teaching and research interests include model-based software development and software testing. He is on the editorial boards of IEEE Transactions on Reliability, Information and Software Technology, Journal of Software Testing, Verification and Reliability, and Software Quality Journal. He was a Program co-chair of ICST 2010 and DSA 2017. He was a general co-chair of MODELS 2009 and Modularity 2015. He is a Senior Member of the IEEE and a member of the ACM.

Indrakshi Ray is a Professor in the Computer Science Department at Colorado State University. She has been a visiting faculty at Air Force Research Laboratory, Naval Research Laboratory, and at INRIA, Rocquencourt, France. She obtained her Ph.D. in Information Technology from George Mason University. Dr. Ray's research interests include security and privacy, database systems, and formal methods for software assurance. She has published over a hundred technical papers in refereed journals and conference proceedings with the support from agencies including Air Force Research Laboratory, Air Force Office of Scientific Research, National Institute of Health, National Institute of Standards and Technology, National Science Foundation, and the United States Department of Agriculture. She is on the editorial board of IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Services Computing, and Computer Standards and Interfaces. She has been a guest editor of ACM Transactions of Information Systems Security and Journal of Digital Library. She was the Program Chair of ACM SACMAT 2006, Program Co-Chair for ICISS 2013, CSS 2013, IFIP DBSec 2003, and General Chair of SACMAT 2008. She has served on the program committees of various conferences including ACM SACMAT, DBSec, EDBT, ESORICS, ICDE, and VLDB. She is a senior member of the IEEE and a member of the ACM.