An Architecture-Oriented Data Warehouse Testing Approach

Neveen ElGamal
Information Systems Department
Faculty of Computers and Information
Cairo University
Egypt
n.elgamal@fci-cu.edu.eg

Ali El-Bastawissy
Information Systems Department
Faculty of Computers and Information
Cairo University
Egypt
aelbastawissy@msa.eun.eg

Galal Galal-Edeen
Information Systems Department
Faculty of Computers and Information
Cairo University
Egypt
galal@acm.org

Abstract

In the past few years, the data warehouse (DW) has regained experts' interest due to the paradigm shift from data storages to data analysis. During the development of DWs data passes through a number of transformations and are staged in multiple storages which might lead to data corruption and/or manipulation. Hence, testing DWs is a vital stage in the DW development life cycle. In this paper, we will present a DW testing approach that is adjustable to fit multiple DW architectures and will present its applicability on three case studies to outline the flexibility and generality of the proposed approach.

1. Introduction

The topic of data warehousing encompasses application tools, architectures, information service, and communication infrastructure to synthesize useful information for decision making from distributed heterogeneous data sources. For this reason, vendors agree that DWs cannot be off-the shelf products but must be designed and optimized with great attention to the customer's situation [20]. Multiple DW life cycle approaches were presented in the literature to discuss how DW systems are built [21, 23]. In those approaches, the architectural design was one of the early and key stages in developing DW systems. On the other hand, testing was not considered in any of the proposed life cycle approaches given that it was always considered in all well-known life cycle approaches like the waterfall and the spiral models. [4, 27]

DW architectural patterns vary from one DW system to another based on user requirements [14]. However, The most common idea in all DW projects is that data is available in one or more data sources and this data needs to be integrated in order to give useful information to assist decision makers to base their decisions on historical behavior of their systems [17].

In the beginning, the data stored in the *Data Sources* (DS) are extracted, transformed, and loaded in the so called *Data Warehouse* (DW). Sometimes this DW is then specialized into a group of business area specific structures each of which contains data that target a specific business area which are called *Data Marts* (DM).

Data passes through several transformations and integration stages before they are loaded from the DSs to the DW or DMs which in most cases force the DW developers to use an intermediary data storage called *Data Staging Area* (DSA); where all the data is transferred to it then transformed and loaded to the DW.

From another perspective, the DW consists of historical data that accumulates years of operational data in one place. Preparing this type of information requires some time, that's why the data stored in the DW are not up to date or even close to that. In some decision making situations, the decision makers want rapid information about data that is not historical, for example; data that is one or two days old. However, they want this type of information to be accumulated from all DSs just like the ones stored in the DW but with less historical dimension. If this type of information is expected to be frequently asked by the decision makers then an *Operational Data Store* (ODS) is required to be part of the DW selected architecture.

Figure 1 shows the most generic and detailed DW architecture that includes most commonly used components and transformations in a DW project. This architecture was proposed under the name of "*Kim-mon Architecture*" which refers to the representation of both Ralph Kimball and Bill Inmon's architectures combined [1]. Data is wrapped from the DSs to the DSA then it travels to the ODS then to the DW then it is specialized into domain specific DMs then finally it reaches the user/decision maker through User Interfaces (UI) for example; OLAP reports, Analysis, and/or DSS tools.

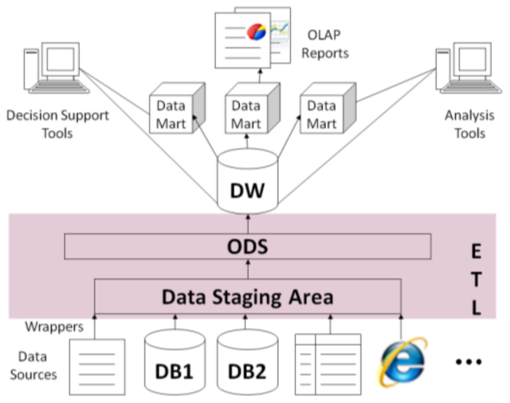

Figure 1. DW Generic Architecture (adapted from [1])

The DW architecture differs from one project to the other based on the specific business requirements. However, the basic component that is available in all DW projects is the DSs. Any other component is included or excluded in the DW project according to the need for it [14]. Variations from the above architecture have been proposed in [19, 14, 21]. These DW architectural patterns simply eliminate or duplicate one of the existing components that are discussed in the Kim-mon architecture. Table 1 summarizes the architectural patterns, discussed in [14], and shows participating components in each.

Table 1. DW Architectural Patterns

Architectural Pattern Name	Architectural Pattern Components
One Layer	DSs→UI
Two Layer	DSs→DW→UI
Independent Data Marts	DSs→DMs→UI
Bus	DSs→DMs→UI
Three Layer	DSs→ODS→DW→UI
Drill-through	DSs→ODS→DW→DM→UI ↓ UI
Hub and Spoke	DSs→ODS→DM→UI
Centralized	DSs→DSA→DW→UI
Federated	DSs→DMs→IL[1]→UI
Kim-mon (Generic)	DSs→DSA→ODS→DW→ DM→UI

Regardless of the architecture of the DW project, data passes through a long way of Extract, Transform, and Load (ETL) processes from its origin in the DSs till it is transformed into information by the UI applications. During this journey data is wrapped, integrated, aggregated, cleansed, loaded, and accumulated which could highly affect the quality of information delivered to the decision makers. Therefore, DW testing is a critical stage in the DW development life cycle which gained multiple researchers' attention to propose a testing technique that is suitable for use in DW projects and provide implementation mechanisms for testing technique to speed the process of testing.

This paper tackles the DW testing from a different perspective. Instead of proposing a testing technique that is suitable for use with a specific DW architecture, this paper proposes a generic DW testing approach and provides an accomodation mechanism that adapts the proposed DW testing approach according to the DW used architecture.

The remainder of this paper will be organized as follows; Section 2 presents a survey on DW testing showing the influence of architectural variations on the DW testing process. Section 3 introduces the generic testing approach that is adequate for use with the Kim-mon architecture. Section 4 describes the technique used to accommodate the proposed testing approach to be adequate for use with other architectures. Section 5 briefly states the implementation details of the accommodation technique. Section 6 discusses findings from applying the proposed technique on several case studies and presents an overall evaluation of the proposed technique. Finally, we conclude our work in section 7.

2. Related Work

Testing DW systems had been studied in literature from various perspectives. Some attempts customized the Software testing strategies to be adequate for use in DW testing [2, 3, 5, 18, 24] while others concentrated on addressing the ETL testing since most of the work is done in the ETL process [7, 35, 22, 26]. A broader view of the DW testing process was studied to address the problem from various perspectives and present to the DW field an integrated solution for DW testing [13, 16, 15, 30, 29, 31, 32, 36, 37, 39, 38]. These approaches were previously studied from the test routine coverage point of view in [11, 10] and it was concluded that none of the existing approaches fully cover the DW testing process. In this paper, we are more concerned with the architectural diversities between the existing approaches and the possibility of generalizing any of the existing approaches to suite several architectures.

By exploring the various DW testing approaches mentioned above, we uncovered considerable diversities between approaches with respect to the architectures that these testing approaches target. From the other perspective, there are many DW architectures defined in literature that needs a DW testing technique to be used in conjunction with them when they are put into operation and yet none of the existing approaches supported these architectures.

[1] IL refers to Integration Layer either Physical or Logical

Table 2 presents a brief comparison allocating existing DW testing approaches to their specified or inferred architectures. The first column displays different architectural patterns, discussed in Table 1, and the second column presents the DW testing approaches along with their architectures that each approach used while describing the DW testing process.

Table 2. Architectural Coverage of DW Testing Approaches

Architecture Name	DW Testing Approaches and Their Used Architectures
One Layer	N/A
Two Layer	[2, 7, 35] DS→DW [24] DS→DW→DM→UI [26] DS→DW→UI
Independent Data Marts	N/A
Bus	N/A
Three Layer	[3] DS→ODS→DW→DM2→ UI [5] DS →ODS→DW→UI
Drill-through	N/A
Hub and Spoke	[15] DS→DSA→DM→UI
Centralized	[22] DS→DSA→DW→DM→UI [29] DS→DSA→DW/DM→DDB3→UI
Federated	N/A
Kim-mon (Generic)	[36] DS→DSA→ODS→DW→ UI

By comparing the architectural components in Table 1 and the architectures in Table 2, we noticed that, few architectures proposed in literature were addressed by the DW testing approaches without further modifications. All the proposed approaches used variations of the defined architectural patterns and customized their DW testing approach based on these variation. It is also shown that some architectures were not addressed by any testing approach like Independent Data Marts, Bus and Federated architectures

What could be concluded from the comparison matrix in Table 2 is that each DW testing approach was defined targeting specific architecture and is therefore adequate for use on DW projects that use the same architecture. If a different architecture is used, then this testing approach will not be adequate for use as it is. Some sort of customization should take place to extend this testing approach to fit the new architecture. This customization could not take place by a testing expert nor a DW expert alone. It is a joint process that should take place benefiting from both experts' knowledge, which is not possible in most cases due to time and budget constraints.

To overcome the weaknesses in the existing approaches, we considered defining a testing approach that is generic enough to be used in multiple DW projects and provide a customization mechanism that is able to accommodate the proposed testing approach to different DW architectures.

Each DW testing approach consists of a group of test routines that describe how this approach tests the DW to improve the quality of the output product. The next section will discuss the group of test routines of the proposed generic DW testing approach.

3. A generic DW testing approach

Test routines defined for DWs are diverse and on different levels of detail, as previously discussed in [10]. To develop a generic DW testing approach that works with different DW architectures, we need to comprehensively determine and describe the complete set of test routines that cover all DW components and transformations. It should also be taken into consideration that these descriptions should be done on a low level of detail to allow later customization for different architectures. For this reason, the Kim-mon architecture presented in Figure 1 will be used for defining DW test routines as it contains all DW components that are most commonly used in all DW architectures.

3.1 Test routine list

Our proposed set of test routines, presented in Table 3, is a refinement of the set of test routines previously presented in [10] when it was used to evaluate and compare the available DW testing approaches. This set of test routines was categorized according to the layer that each test routine targets, the level of detail that this test involves, and when this test takes place. It is worth mentioning that the User Interface layer (DM→UI) will not be part of our proposed solution. The refinements took place to come up with a uniform and consistent set of test routines. These refinements are as follows:

1. Unifying synonymous test routines like Field mapping, Data type compatibility, and Data Layout Format.
2. Removing the *Overall* test routines and define them redundantly on each layer.

As shown in Table 3, the rows represent the layers of the Kim-mon architecture, the columns represent the level of detail that each test routine involves, and test routine periodicity is represented by *italicizing* test routines that are conducted after system development. The underlined test routines are the ones that are redundant on several layers. Introducing *redundant test routine* came from the need to support multiple architectures. When a different architecture is under test the proposed approach will customize the Table 3 to fit the new architecture.

After identifying the set of test routines that are suitable for use in DW projects, it is mandatory to provide the tested with descriptions of these test routines to assist him/her during the testing process. The next section

[2] *Italic* data warehouse components in Table 2 refer to components in the data warehouse architecture that are defined in the proposing approach but not tested.

[3] DDB refers to Dimensional Database

presents the description scheme that we introduced and used to provide descriptions for all test routines inclosed in the test routine list.

Table 3. Proposed DW Test Routines

	Schema	Data	Operation
DS→DSA	1. User requirements 2. Field mapping a. Field naming, b. Data types match, c. Field size match, 3. Correct data selection	*1. Record counts* 2. Threshold test 3. Data boundaries 4. Data profiling *5. Random record comparison* 6. *Field to field comparison*	*1. Rejected records* 2. Data access 3. Security
DSA→ODS	1. Schema Design 2. Field mapping a. Field naming, b. Data types match, c. Field size match, d. Data type constraints 3. Aspects of transformation rules a. Captured b. Formula syntax c. Transformation Logic	*1. Record counts* 2. Data integrity a. Identity integrity b. Referential integrity c. Cardinal integrity d. Inheritance integrity e. Domain integrity f. Relationship dependency integrity g. Attribute dependency integrity 3. Parent-child relationship *4. Duplicate detection* 5. Threshold test 6. Data boundaries *7.* Data profiling *8. Random record comparison* *9. Field to Field Comparison* *10. Surrogate keys* *a. Correctness* *b. Integrity*	1. Review job procedures 2. Error logging 3. Performance *4. Rejected record* 5. Data access 6. Forced Error test 7. Stress test 8. Security
ODS→DW	1. User requirements coverage 2. DW conceptual schema: a. Conformed hierarchy b. Understandability c. Usability d. Mapping to logical model 3. DW logical model: a. Mapping to physical model b. Functionality c. Performance (comply with MDNF) 4. Integrity constraints 5. Hierarchy level integrity 6. Granularity 7. Derived attributes checking 8. Field mapping a. Field naming, b. Data types match, c. Field size match, d. Data type constraints	*1. Record counts* 2. Data integrity a. Identity integrity b. Referential integrity c. Cardinal integrity d. Inheritance integrity e. Domain integrity f. Relationship dependency integrity g. Attribute dependency integrity 3. Parent-child relationship *4. Duplicate detection* 5. Threshold Test 6. Data boundaries 7. Data profiling *8. Random Record Comparison* *9. Field to field comparison* 10. No constants loaded 11. No Null records loaded 12. Simulate data loading 13. Data aggregation 14. Reversibility of data from DW to DS *15. Confirm all fields loaded* 16. *Data freshness*	1. Review ETL documentation 2. ETL test a. ETL activity ordering b. ETL recoverability c. Job sequence d. Error propagation through jobs e. Job resetting f. Batch failure propagation g. Batch reset in case of failure 3. Scalability 4. Initial load *5. Incremental load* 6. Data access *7. Rejected record* 8. Performance 9. Error logging 10. Forced error test 11. Stress test 12. Security 13. HW and SW configuration
DW→DM	1. Schema Design 2. Calculated members 3. Irregular hierarchies 4. Correct data filters 5. Additivity guards	1. Measure Aggregation	1. Security 2. HW and SW configuration

3.2 Test routine description scheme

Test routines listed in Table 3 are a group of test routines that are refined and few of them were introduced to the DW testing process. To be able to use these test routines they need to be fully described. The full description will not appear in this section. However, The description scheme that we introduced to broadly describe all test routines is as follows:

1. Name: The common name used in testing field for this test routine.
2. Layer: Which layer of the Kim-mon architecture does this test routine take place?
3. Level: What is being tested in this test routine? (Schema/Data/Operation)
4. Objective(s): a textual description of the test routine showing its objective(s).
5. Type: The type of the test routine: (Verification/Validation)
6. Severity: The importance of this test routine (Mandatory, Recommended, Optional)
7. Periodicity: How often does this test routine take place? (Schema Change/Data Load)
8. Part Under Test: which part of the component under test is being tested (ex: Schema, Table, Attribute, etc…)
9. Input(s): Required documents that need to be available to conduct this test routine (if any).
10. Testing Scenario: The detailed description of how this test routine is conducted.
11. Automation: the possibility of automating this test routine and the type of automatic assistance required, (Testing Tool, Data Generation Tool, Test Case Generation Tool)

Each test routine was described using the above scheme. Required information about each test routine was gathered from existing testing approaches and few of them were defined from scratch. For example; the test routine named *Duplicate Detection* is described in Table 4 using the aforementioned scheme.

Table 4. Duplicate Detection Test Routine Description

Name: Duplicate Detection
Layer: DSA→ODS
Level: Data
Objective(s): **C**onfirm that no duplicate records exist in the ODS
Type: Verification
Severity: Mandatory
Periodicity: Data Loads
Part Under Test: Every Table in the Destination
Input(s): None
Testing Scenario:

> There are two types of duplicated that needs to be detected and resolved:
>
> I. Duplicates resulting from incorrect data transformation procedure.
> II. Duplicates resulting from integrating data from different data sources.
>
> This first type of duplicates could be detected as follows:
>
> 1. Run a query on each destination table to retrieve duplicates. An example of this query could be as follows:
>
> ```
> Select *
> From <TableName>
> Group by <AllAtributes>
> Having count(*) >1.
> ```
>
> 2. If this query returns any results, it means that there are duplicate records in this table and these duplicates are a result of an incorrect transformation process
>
> The second type of duplicates could be detected by applying one of the duplicate detection techniques that have been severely studied in science to solve the problem of duplicate detection and resolution in integrated data. [12]

Automation: Testing Tool is required

All test routines displayed in Table 3 were described using the same scheme, used in Table 4, to provide the testers with some sort of instruction manual for DW testers, available at [8, 9], supplying them with any required information regarding the process of DW testing. However, this test routine description is adequate for use with the Kim-mon architecture only. If another architecture is used in a DW project, these test routines need to be adapted to be adequate for use with the used architecture. This paper proposes a customization mechanism in the next section.

4. Multiple architectural accomodation

As it was previously discussed in section 1, DW components are the interrelated parts of the DW architecture that are connected together to transform data in DSs into information. Moreover, by studying the available architectural patterns discussed in literature, it was notable that all the architectural components are always used in the same order (DS, DSA, ODS, DW, DM) if they are part of the selected architecture.

Since we are concerned with testing DWs with different architectures, then the above mentioned set of test routines that are defined on the Kim-mon architecture need to be customized in a way to fit different architectures.

Each test routine stated in Table 3 is mapped to a specific DW layer. Each layer consequently relates this test routine to two DW components (source and destination components). For example, the test routine "Duplicate Detection" presented in Table 4 is defined on the DSA→ODS layer, hence, relates this test routine to both layers the DSA and ODS. However, this test is concerned with detecting duplicates that exist between the ODS records and it is not concerned with the DSA by any means. On the Contrary, the test routine "Record Counts", defined on the DS→DSA layer in Table 3, requires the participation of both the DS and the DSA in the test routine in order to compare record counts and confirm that they are matching.

So, relating test routines to DW layers only will not help in the process of test routine customization to multiple architectural patterns because each test is not related explicitly to each DW component. For this reason, adding more descriptive attributes to the test routine description scheme discussed in the previous section is needed to include in the test routine description the prerequisite components that this test routine involves or requires.

Two prerequisite attributes need be specified for each test routine; one is a source prerequisite and the other is a destination prerequisite. Depending on the objective of each test routine and the role of the DW component with respect to the test routine. Whether it is the source of the data being transformed, or the component that receives the data. These two attributes are not mandatory to all test routines. Some test routines might require both attributes to be specified like the Record Counts test routine discussed above because its objective is to compare results between the source and destination. While another test routine requires only one prerequisite, either the source or the destination, because they check one component's consistency or its validity with respect to some other parameter like user requirements or business rules. Example for these test routines is the "Duplicate Detection" and "User Requirements". The two attribute templates are as follows:

Prerequisite(s):

- Source: <ComponentName>
- Destination: <ComponentName>

From another perspective, Test routines stated in Table 3 could be clustered according to the purpose of each. Some of them are concerned with the successful transformation of data from the data sources through all the DW system data storages till it reaches the user. Examples of these test routines are Record Counts, Duplicate Detection, Data Boundaries, Error Logging, etc. While others are concerned to experiment a specific functionality that is served by a specific DW component. For example, DW conceptual schema, DW Logical Schema, and Measure Aggregation. This type of clustering needs to be taken into consideration while defining each test routine, because when a DW component is not part of the architecture used, then these test routines need not be considered in the proposed test routine list. For this reason, an extra attribute named "Single Layer Test (SLT)" is assigned the value 1 for test routines that experiment specific functionalities to be able to differentiate between the purpose of each test routine. The template of the SLT attribute is as follows:

SLT: <Binary>

Till this point, all test routines have been mapped to its proper components and all desired information about each test routine is available in the test routine description over the Kim-mon architecture. But, when the architecture under test is a different architecture, a mapping technique needs to be defined to re-direct the test routines that refer to a DW component that is not part of the architecture under test to another component that is present in the given architecture. This technique is defined as follows;

Using the extra two attributes that define the prerequisites, each test routine is related to one or more DW component each of which acts as a source or destination prerequisite. When the test routine is related to a prerequisite DW component that is not part of the architecture under test, alternative component needs to take over the place of the absent component and this test routine will be conducted on the alternative component instead to guarantee a proper transformation of data between participating DW components. This rule does not apply to test routines whose attribute SLT takes the value 1 since these test routines are testing a specific functionality of the prerequisite component that is currently not part of the architecture used. Therefore, no alternative component will perform this specific functionality and consequently the test routines testing it need not be taken into consideration.

Choosing the suitable alternative component is a decision that is taken based on the type of the absent component, whether it was a source or destination prerequisite with respect to the test routine. If it was a source prerequisite, the preceding alternative needs to be chosen as the absent component's replacement, and if it was a destination prerequisite, the succeeding alternative will be chosen. Figure 2 presents the succeeding and preceding alternatives for all possible DW components.

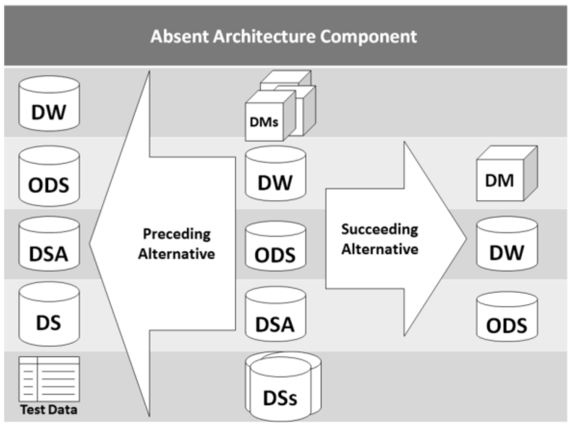

Figure 2. DW Component Alternatives

It is quite common in the architectures discussed earlier in section 1 that more than one consecutive component could be absent from the DW architecture with respect to the generic (Kim-mon) architecture presented in Figure 1. To find a suitable alternative for test routine's prerequisites, transitivity is applied on the alternative relationship defined above. The relationship is transitive in the sense that if an absent component is a preceding or a succeeding alternative of another absent component, then find the alternative component's alternative to replace it and assign it to the specified test routine.

To determine the alternatives for each absent prerequisite the following steps will take place:

1. For each test routine that is not a Single Layer Test, determine whether its prerequisites are absent or present components. Since Single Layer Tests need not be mapped on any other layers because they target functionalities that are specific to their prerequisites and when these prerequisites are not part of the architecture under test, these test routines will not be part of the customized test routine list.
2. For each absent prerequisite, determine its preceding or succeeding alternative components, depending on the type of the prerequisite relation whether this component is a source or a destination prerequisite to this test routine.
3. For each preceding alternative component, if it is also an absent component, then get its preceding alternative. Repeat this sequence until the transitivity rule leads to an alternative that is not an absent component.
4. For each succeeding alternative component, if it is also an absent component, then get its succeeding alternative. Repeat this sequence until the transitivity rule leads to an alternative that is not an absent component.

This technique will assign test routines to DW components that are part of the architecture under test to be able to test it properly.

4.1 Example

If the architecture used for the DW is the Two Layer architecture, presented in Table 1, which consists of DSs, DW, and DMs. Then the two components DSA and ODS are considered absent components with respect to the generic (Kim-mon) architecture presented in Figure 1.

By applying the test routine customization on the test routines presented in Table 3 and re-directing test routines to their suitable prerequisite alternatives as discussed previously the outcome of this process will result in the set of test routines presented in Table 5 that was customized to the Two Layer architecture not by removing the two layers DS→DSA and DSA→ODS but redirecting their test routines to the appropriate alternative layers.

Table 5. Test Routines for the Two-Layer Architecture

	Schema	Data	Operation
DS→DW	1. User requirements 2. Field mapping a. Field naming, b. Data types match, c. Field size match, d. Data type constraints 3. Correct data selection 4. Schema Design 5. Aspects of transformation rules a. Captured b. Formula syntax c. Transformation Logic 6. User requirements coverage 7. DW conceptual schema: a. Conformed hierarchy b. Understandability c. Usability d. Mapping to logical model 8. DW logical model: a. Mapping to physical model b. Functionality c. Performance (comply with MDNF) 9. Integrity constraints 10. Hierarchy level integrity 11. Granularity 12. Derived attributes checking	1. Record counts 2. Threshold test 3. Data boundaries 4. Data profiling 5. Field to field comparison 6. Random record comparison 7. Parent-child relationship 8. Duplicate detection 9. Surrogate keys a. Correctness b. Integrity 10. Data integrity a. Identity integrity b. Referential integrity c. Cardinal integrity d. Inheritance integrity e. Domain integrity f. Relationship dependency integrity g. Attribute dependency integrity 11. No constants loaded 12. No null records loaded 13. Simulate data loading 14. Data aggregation 15. Reversibility of data from DW to DS 16. Confirm all fields loaded 17. Data freshness	1. Rejected records 2. Data Access 3. Security 4. Review job procedures 5. Error logging 6. Performance 7. Forced Error test 8. Stress test 9. Review ETL documentation 10. ETL test a. ETL activity ordering b. ETL recoverability (Robustness) c. Job sequence d. Error propagation through jobs e. Job resetting f. Batch failure propagation g. Batch reset in case of failure 11. Scalability 12. Initial load 13. Incremental load 14. HW and SW configuration
DW→DM	1. Schema Design 2. Calculated members 3. Irregular hierarchies 4. Correct data filters 5. Additivity guards	1. Data Aggregation	1. Security 2. HW and SW configuration

5. Implementation

To structure and keep track of this large amount of information required for test routines and the relationships between test routines and their prerequisite components, It was mandatory to store these details in a structured yet flexible format to accommodate any future changes that might take place; like adding test routines, modifying test routine descriptions, and/or deleting unnecessary test routines.

The proposed test routine description and customization mechanisms were implemented using the graph database Neo4j [25]. We chose the graph database because of its flexible structure that could evolve through time without affecting stored information. It is also distinguished to have a very simple yet powerful querying language called *Cypher* that is used as both data definition and data manipulation language.

Using Cypher query, we have prepared a data definition script that fully defines and fills the graph database with all test routine definitions and relationships. Another Cypher query template was defined to accommodate the graph database according to the architecture under test. Finally, a third Cypher query was defined to generate from the customized graph database a detailed test routine list clustered according to levels and layers as the one displayed in tables Table 3 or Table 5 when the architecture under test is the Kim-mon architecture or the Two-Layer Architecture, respectively.

6. Case studies and evaluation

The proposed approach claims to provide a testing technique that is adjustable according to the architecture of the DW system under test. For this reason, it was mandatory to experiment with it using different case studies with different DW architectures.

The experimentation mechanism we used to apply the proposed customization technique was getting access to abstract information about the DW under test, proposing the set of test routines adequate for the given architecture, and getting the feedback from the DW testers regarding the proposed test routine list. On the other hand, we considered studying the testing technique used in the DW under test (if available) and compared it with our proposed test routine list.

During the selection of the case studies, we were keen to find case studies with different architectures and to choose companies in different sizes. The proposed approach was applied to three case studies from three different sized companies;

1. CentriVision: a small sized Egyptian company, founded in 2003, whose development services involves business intelligence solutions.[6]
2. SMSMT: a medium sized Australian company, founded in 1986, that provides testing as one of its services.[28]
3. Teradata: a large sized American company, founded in 1979, that sells analytic data platforms, applications and related services. [34]

Each of these companies was using a different DW architecture in the DW project they supplied us with, except for Teradata since it uses a generic architecture for all its projects. The architectures of the three case studies are displayed in Figure 3.

The results concluded from applying the proposed accommodation mechanism to customize a test routine list for each case study is presented separately in the following three sections. Each section will present a comment on the adequacy of the proposed approach when applied to one of the case studies.

6.1 CentriVision

According to the DW development/testing team at CentriVision, testing in this DW project was conducted using team members' experiences. There was no standard testing technique used. However, a set of tests takes place at different levels of detail to guarantee the quality of the DW under development. The categorization of CentriVision's undocumented testing activities is displayed in Table 6 categorized by layers and levels they apply to with respect to the DW architecture used.

Table 6: CentriVision Test Routine Categorization

Level / Layer	Schema	Data	Operation
DS→DM	• User Requirements Coverage • DM Schema Design • Field mapping in Transformation rules	• Record Counts • Data Aggregation • Calculated members • No null measures exists • Duplicate detection • Simulate data loading • Data freshness	• HW and SW configuration • Scalability of data • Performance Test • Review Job Procedures
DM→UI	User requirements coverage	-	-

By communicating the proposed test routine list with a project manager at CentriVision the feedback was as follows:

1. Most recommended test routines that were proposed in the test routine list are usually conducted either directly by snooping for mismatches in the migrated data or indirectly during the process of defect tracking.
2. Supplying them with the customized test routine list is of great help to give them ideas about what needs to be tested and how these tests could be conducted rather than depending on tester's experience and jeopardize the DW quality.
3. Regarding the test routines they did not support, their feedback was that it would highly increase the quality of their output products if taken into consideration during the testing process.

From our point of view, depending on the tester's experience in testing the system is not a reliable way of finding errors. Having a consistent and well documented testing strategy that could be used as an instruction manual for the tester to follow during the testing process could highly assist the tester in knowing the possible vulnerabilities that could take place in the DW and testing the system to prevent it against possible threats. Applying the testing technique that is defect oriented; where the tester follows defects to fix the errors, is not the best way to find and fix errors. It could be possible that errors exist in the system, but the right test was not conducted to reveal them.

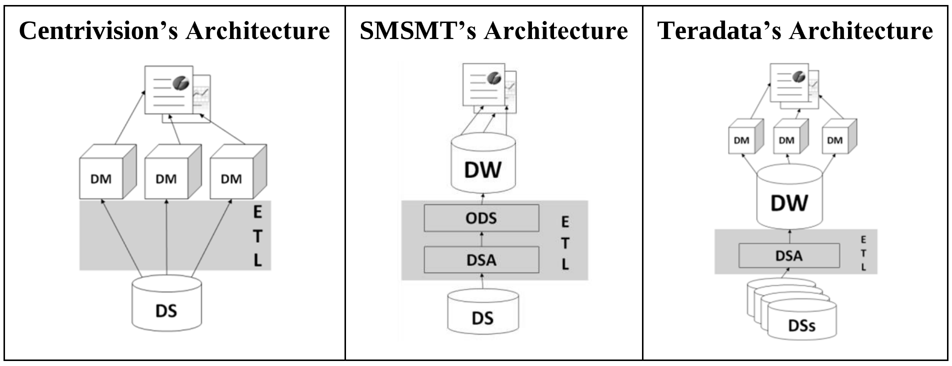

Figure 3. Architectures of the Case Studies

6.2 SMSMT

According to a Testing and Quality Assurance consultant at SMSMT, the company agreed to use a set of test routines in their current project of developing a DW for a Legacy system. Table 7 presents the test routines used in this case study. The set of test routines used in this case study are mostly data tests which involve simulating and comparing data transformations between different data storages.

Table 7. SMSMT Test Routine Categorization

Level / Layer	Schema	Data	Operation
DS→DSA	Field Mapping	• Not nullable fields • Record counts • Field to field comparison • Simulate data loading	Delta load test
DSA→ODS	• Field mapping • Transformation rules test	• Record counts • Field to field comparison • Simulate data loading • Domain Key Metric test	Delta load test
ODS→DW	• Field mapping • Transformation rules test	• Record counts • Field to field comparison • Simulate data loading • Domain Key Metric test	• Initial load test • Incremental load test
DW→UI	-	• Result comparison across data sources	-

By communicating the proposed set of test routines customized for the SMSMT case study, the feedback of the test and quality assurance consultant was as follows:

1. The proposed set of test routines is quite comprehensive for a one-time load, however, it misses the tests for delta loads on several layers.
2. Some test routines that we proposed needs to be conducted on different layers like the performance and stress tests needs to take place between DS and DW not only ODS and DW.
3. The proposed set of test routines lacks the consideration of timeliness of test routines to guard against the execution of test routines that might cause conflicts in the data if run at the same time.

The comparison between the proposed and the used set of test routines showed that 90% of the test routines used in SMSMT case study are included in the proposed set of test routines which confirms the soundness of the proposed approach and proves its adjustability to a different architecture.

6.3 Teradata

According to the interview conducted with the test manager at Teradata Egypt, there exists a documented, well defined, and formulated testing strategy that is used in all DW projects in Teradata, yet it is still in the editing phase [33]. The basic modules of tests were interpreted from the testing strategy and was categorized according to layers and levels of the DW architecture. Table 8 presents the test routines introduced in the Teradata testing strategy.

As it is shown in Table 8, all test routines used by Teradata are tests that take place before system delivery. Any post delivery tests are conducted by the quality assurance team based on the customer's reporting of errors or inconsistencies on the output data. The Teradata testing strategy is also notable for its coverage on all three testing levels (Data, Schema, and Operation), which highly enriches the quality of their DW products.

Table 8. Teradata Test Routine Categorization

Level / Layer	Schema	Data	Operation
DS→ DSA	• User Requirements • Data types • Columns order	• Record counts • *Pattern counts* • Data profiling • Random record comparison	• *All sources are loaded* • Rejected records
DSA→DW	• Field mapping • Review Logical schema for customizations • *Naming Conventions*	• Record Counts • Field to field comparison • Random record comparison • Primary key index integrity • Surrogate keys integrity and correctness • Referential integrity • Domain integrity • *History integrity* ○ *Reverse History* ○ *History Overlap* ○ *Null History* ○ *History Gap* ○ *Open end History* •	• Rejected records • Performance test • *ETL Scheduling*
DW→DM	• Schema Design	• Record counts • Field to field comparison • Data aggregation • *Business Rules Integrity*	• Security • HW and SW configuration
DM→UI	• User Requirements Coverage	• Business logic testing	

A comparison took place between the Teradata testing strategy and the proposed test routine list customized on the Teradata architecture. Table 9 presents a numerical reference, for the number of test routines that Teradata testing strategy takes into account from the proposed test routine list. For example, On the DSA→DW layer at the Data level, the Teradata testing strategy considers 8 of the proposed 17 test routines and applies an extra 5 test routines to test this layer of the DW.

This comparison revealed that, in general, the proposed test routine list agrees with the Teradata testing strategy in 40% of the proposed test routines. This is because the proposed set of test routines contains all possible test routines that could be used to test any DW, however; the proposed test routines should not be all conducted on any system. Some of them are alternative to each other which leaves to the tester the option of choosing among them.

Table 9. Teradata Vs Proposed Testing Strategy

Level / Layer	Schema	Data	Operation
DS→ DSA	1/4	3/6 +1	1/3 +1
DSA→DW	2/9 +1	8/17 +5	7/14 +1
DW→DM	1/5	1/1 +1	1/2

According to the analysis abstracted in Table 9, it is clear that the Teradata testing strategy agrees with the proposed test plan to a great extent on the *data* level comparisons. On the *schema* level, however, the Teradata testing strategy lacks the support of most of these types of tests. This was due to their reliance on the data tests that will reveal any possible schema or structural problems or inconsistencies.

What Table 9 revealed as well was that the tests on the *operational* level like scalability, security, or stress were not conducted on a project basis. This is because they use the Teradata Database Management System which is extensively tested from these perspectives and results are guaranteed if proper HW and SW configurations took place.

On the other hand, to fairly compare the two testing strategies, Table 9 shows that on certain levels and layers the Teradata testing strategy supported some test routines that were not part of the proposed test routine list. These test routines are *Italicized* in Table 8. By studying these test routines we see that including these test routines in our proposed testing technique would be a good addition to it.

From a different angle, after communicating the proposed detailed test routine list with the Test Lead and a Testing Developer at Teradata Egypt, we were able to get their feedback regarding the proposed set of test routines. Their comments were as follows:

1. The proposed set of test routines could be considered as the standard suite of tests required for system test a DW solution.
2. It is satisfactory for the technical requirements of testing a DW solution.

From their point of view, what lacks the proposed testing strategy is reference to commercial tools that could be used to automate each test routine, showing stakeholders involvement in each test routine and absence of the test routines supported by Teradata and not included in the proposed testing strategy as discussed previously.

In spite of the above drawbacks, due to the flexibility of the graph database, they could be easily overcome by introducing the required modifications on the graph database with minimal change in the Cypher query graph creator script.

6.4 Overall evaluation

As discussed previously in section 02 the main drawback of existing testing approaches was their rigidity with respect to the architecture of the DW. Each approach assumed that the architecture used in their approach is the DW architecture mostly used and proposed a testing strategy for it. What distinguishes our proposed approach is its flexibility of adapting the test routines according to the architecture under test unlike other approaches.

Table 10 presents possible architectures that the proposed test routine list could be customized for. The proposed approach was the first approach that covers testing DWs with different architectures and not only supported well-known architecture types discussed in the literature, but also supported other DW architectures that are sometimes used but it was not named in the literature.

Table 10. Proposed Framework Architectural Coverage

Architectural Pattern Name	Proposed Approach's Coverage
One Layer	X
Two Layer	√
Independent Data Marts	√
Bus	√
Three Layer	√
Drill-through	√
Hub and Spoke	√
Centralized	√
Federated	X
Kim-mon (Generic)	√
DS→DSA→DW→DM	√
DS→DSA→DM	√

Architectures not supported in the proposed approach are the ones that involve a special nature layer which is not commonly used in DWs. For example, the Single Layer architecture integrates data virtually through a group of on-the-fly transformation rules. Neither integrated data are stored nor are historic data available for any decision making processes. This type of DWs requires a custom made test plan that could not be provided given the proposed test routine list. The same rule applies for the Federated architecture, where the Integration Layer is a special layer that needs a custom made testing technique. However, our proposed test routine list is adequate for testing the transformation of data from the DSs to the federated DMs smoothly.

What needs to be clarified regarding the architectural coverage is that the proposed approach does not present test routines that targets the layer of the UI which is a drawback that affects all supported architectures. However, The UI Layer testing were beyond our scope and excluding it was based on expert's recommendation because there are different means of UIs like reports, charts, DSS tools, and Analytical tools; where, each of which required a different testing technique for their verification and validation.

7. Conclusion and future work

What distinguishes this study from any other proposed solution for the issue of DW testing is the comprehensiveness of the description scheme that was used to describe all test routines, studying DW architectures to interpret means of relating architectural components in order to fulfill the aim of providing the DW testers with a generic solution for DW testing adequate for use in multiple projects with different architectures, benefiting from other people's work by integrating the proposed solution with their work to come up with a detailed technique for DW testing with minimum effort and maximum gain, and finally, considering future system modifications by using a graph database to store the test routine descriptions to be easily extended to contain various system additions or changes.

Future work in this area could be categorizing test routines according to the well known software testing phases, namely; Unit testing, Integration testing, System testing, and User Acceptance testing. Since it has been widely agreed upon that the testing phase is categorized into the aforementioned test phases, it would be of great help to the testing team to suggest for them test routines given in the categorization scheme they are familiar with.

Another possible extension to the proposed work could be including the group of test routines targeting the layer of User Interfaces since it has been skipped, though our research.

Test routines suggested in this paper are all possible test routines. Not all of them need to be conducted on any DW to test it. Consequently, testers are given the opportunity to choose the proper set of test routines to conduct on their DW. Hence, providing the testers with enough information about relationships between test routines and possible dependencies that might take place between them might help them choose the proper set of test routines without sacrificing their system's quality.

In the end, we would like to conclude that the proposed architecture-oriented testing approach may not be the ultimate solution for DW testing, however, it has gained multiple user's trust. It is powerful for its flexibility and might be adequate for use by small to medium sized DW development companies that do not have standardized or comprehensive DW Testing frameworks.

8. Acknowledgements

We would like to thank the companies who supported us with time and resources to be able to experiment with the proposed architecture-oriented DW testing approach (Teradata Egypt, Centrivision, and SMSMT).

9. REFERENCES

[1] R. Abellera, *Data Warehouse Architectures: Overview of the Corporate Information Factory and the Dimensional Modeling*, The Data Warehouse Institute, 2010.

[2] C. Bateman, *Where are the Articles on Data Warehouse Testing and Validation Strategy?*, *Information Management*, 2002.

[3] S. Bhat, *Data Warehouse Testing - Practical*, *Stick Minds*, 2007.

[4] B. Boehm, *A Spiral Model of Software Development and Enhancement*, 21 (1988), pp. 61-72.

[5] K. Brahmkshatriya, *Data Warehouse Testing*, *Stick Minds*, 2007.

[6] CentriVision, www.Cenrivision.com, 2003.

[7] R. Cooper and S. Arbuckle, *How to Thoroughly Test a Data Warehouse*, *Software Testing Analysis and Review (STAREAST'02)*, Orlando, Florida, 2002.

[8] N. ElGamal, *Data Warehouse Test Routine Descriptions*, *Technical Report*, www.researchgate.net/publication/289729988_Data_Warehouse_Test_Routine_Descriptions, 2016.

[9] N. ElGamal, *Data Warehouse Testing*, *PhD. Thesis*, *Faculty of Computers and Information*, Cairo University, 2015, Appendix B, pp. 193-228.

[10] N. ElGamal, A. ElBastawissy and G. Galal-Edeen, *Data Warehouse Testing*, *Proceedings of the Joint EDBT/ICDT PhD Workshop*, ACM, Genoa, Italy, 2013.

[11] N. ElGamal, A. ElBastawissy and G. Galal-Edeen, *Towards a Data Warehouse Testing Framework*, *Proceedings of the 9th International Conference on ICT and Knowledge Engineering (ICT&KE'11)*, IEEE, Bangkok, Thailand, 2011, pp. 67-71.

[12] A. K. Elmagarmid, P. G. Ipeirotis and V. S. Verykios, *Duplicate Record Detection: A Survey*, IEEE Transactions on Knowledge and Data Engineering, 19 (2007), pp. 1-16.

[13] M. Golfarelli and S. Rizzi, *A Comprehensive Approach to Data Warehouse Testing*, *Proceedings of the ACM 12th international workshop on Data warehousing and OLAP (DOLAP'09)*, ACM, Hong Kong, China, 2009, pp. 17-24.

[14] M. Golfarelli and S. Rizzi, *Data Warehouse Design: Modern Principles and Methodologies*, McGraw Hill, 2009.

[15] M. Golfarelli and S. Rizzi, *Data Warehouse Testing*, International Journal of Data Warehousing and Mining, 7 (2011), pp. 26-43.

[16] M. Golfarelli and S. Rizzi, *Data Warehouse Testing: A prototype-based methodology*, Information and Software Technology, 53 (2011), pp. 1183-1198.

[17] J. Guerra and D. Andrews, *Why You Need a Data Warehouse*, www.rapiddecisions.net, Copyright Andrews Consulting Group, Inc., 2011.

[18] S. L. Gupta, P. Pahwa and S. Mathur, *Classification of Data Warehouse Testing Approaches*, International Journal of Computers and Technology, 3 (2012), pp. 381-386.

[19] W. H. Inmon, *Building the Data Warehouse*, Wiley Comp., 1996.

[20] M. Jarke, M. Lenzerini, Y. Vassiliou and P. Vassiliadis, *Fundamentals of Data Warehouses*, Springer-Verlag New York, Inc., 2001.

[21] R. Kimball, L. Reeves, W. Thornthwaite and M. Ross, *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses*, John Wiley \&amp; Sons, Inc., 1998.

[22] M. P. Mathen, *Data Warehouse Testing*, *Infosys Technologies Limited*, 2010.

[23] R. Mattison, *The Data Warehousing Handbook*, XiT Press, Oakwood Hills, Illinois-USA, 2006.

[24] A. Munshi, *Testing a Data Warehouse Application*, *Wipro Technologies*, 2003.

[25] Neo4j, *Neo4j Graph Database*, 2013.

[26] V. Rainardi, *Testing your Data Warehouse*, *Building a Data Warehouse with Examples in SQL Server*, Apress, 2008.

[27] W. W. Royce, *Managing the development of large software systems: concepts and Techniques*, *Proceedings of the 9th International Conference on Software Engineering (ICSE'87)*, Monterey, California, USA, 1987, pp. 328-338.

[28] SMSMT, *SMS Management and Technology* *www.smsmt.com*, 1986.

[29] P. Tanuška, O. Moravčík, P. Važan and F. Miksa, *The Proposal of Data Warehouse Testing Activities*, *Proceedings of 20th Central European conference on Information and Intelligent Systems*, Varaždin, Croatia, 2009, pp. 7-11.

[30] P. Tanuška, O. Moravčík, P. Važan and F. Miksa, *The Proposal of the Essential Strategies of Data Warehouse Testing*, *Proceedings of 19th Central European Conference on Information and Intelligent Systems (CECIIS'08)*, 2008, pp. 63-67.

[31] P. Tanuška, P. Schreiber and J. Zeman, *The Realization of Data Warehouse Testing Scenario*, *proizvodstvo obrazovanii. (Infokit-3) Part II: 3 meždunarodnaja nature-techničeskaja konferencija.*, Stavropol, Russia, 2008.

[32] P. Tanuška, W. Verschelde and M. Kopček, *The proposal of Data Warehouse Test Scenario*, *Proceedings of European conference on the use of Modern Information and Communication Technologies (ECUMICT'08)*, Gent, Belgium, 2008.

[33] Teradata, *Teradata Testing Strategy*, 2014.

[34] Teradata, *www.teradata.com*, 1980.

[35] J. Theobald, *Strategies for Testing Data Warehouse Applications*, *Information Management*, 2007.

[36] D. Vucevic and W. Yaddow, *Testing the Data Warehouse Practicum - Assuring Data Content, Data Structures and Quality*, Trafford, 2012.

[37] D. Vucevic and M. J. Zhang, *Testing Data Warehouse Applications*, Trafford Publishing, 2011.

[38] W. Yaddow, *Conducting end-to-end testing and quality assurance for data warehouses*, *IBM Data Magazine*, 2013.

[39] W. Yaddow, *Enriching data warehouse testing with Checklists*, *IBM Data Magazine*, 2013.