Coverage Criteria for Testing SQL Queries

Mona Gharib Zahraa E. Mohamed Mohamed Reda

Faculty of Science, Mathematics Department, Zagazig University

Zagazig, Egypt

Abstract—Database query language could be difficult to non-expert users and learning formal queries takes a lot of time. Accessing a database requires machine-readable instructions that not everybody is supposed to know, they should be able to ask a question in natural language without knowing either the underlying database schema or any complex structured machine language. Also, we can type a question or a sentence in their natural language. We will use mapping between natural language and the database SQL query. In this paper, we will use the coverage criteria for evaluating the adequacy of a test suite for SQL queries that retrieve information from the database that join information from different tables and whish, selected data is further processed. We will use automatic mapping of questions into SQL queries. We apply test case on AdventureWorks2012 database and use different condition coverage criteria for representing all possible combinations and results of evaluation conditions for a SQL query then coverage criteria used to develop test inputs queries from a real-life application.

Index Terms—SQL Testing, Test Adequacy Criteria, Coverage Criteria, Natural Language Interface.

INTRODUCTION

Ranging from legacy applications in use in the banking, financial or insurance sectors to modern e-commerce applications, there is one component which they all have in common, the database, where sensitive business information is stored and retrieved. Programming languages have experienced a paradigm shift from monolithic programs written in old imperative languages to highly scalable enterprise applications, reusable components and web services written in object-oriented languages. At the same time, database management systems (DBMS) have evolved, increasing their performance, scalability and reliability.

A major problem that faces the NLIDB designer is the identification of the tables that contain the required information and the desired attributes in query. In [1], previous work used static built-in templates of possible production rules for the possibly introduced queries.

The standard approach to database NLP systems relies on creating a 'semantic grammar' for each database, and uses it to parse the NL questions. The semantic grammar creates a representation of the semantics of a sentence. After some analysis of the semantic representation a database query can be generated to SQL. The most frequently used SQL statements in commercial applications can retrieve information (SELECT queries) [2], that use a common set of major characteristics, such as the database schema and the core clauses for projecting, joining, selecting and grouping data. However, developing a single statement may be a test database and/or calling the query with different parameters. complicated task and queries using GROUP BY, ORDER and HAVING clauses are considered especially difficult by programmers. The aim of this work is to define coverage criteria for assessing the adequacy of the test suite to exercise various situations that affect the data retrieved by SQL query.

Our approach studies queries in an isolated way without considering the imperative code where they will be embedded and the tests can be used as prerequisites for embedding queries in the imperative code. This paper improves the approach that given in [3], where queries only had FROM and WHERE clauses and conditions were exclusively composed of attributes, constants or NULL. Moreover, we consider parameters, GROUP BY and HAVING clauses, aggregate functions, ALL and DISTINCT quantifiers along with UNION operator. The approach involves building one or more coverage nodes that are created on the basis of the structure query and database schema. Nodes are arranged in trees for assessing the adequacy of join and selection operations, and in sets for assessing the coverage of the processing performed after selection. Then, coverage is evaluated in relation to the load provided by the test database and to the actual parameters dependent on the imperative code. After evaluating coverage, with the information of the non-covered situations in the nodes, the tester has guidelines to follow in the process of completing the test suite by adding or changing information in the test database, creating a new

The paper is organized as follows: in section 2 introduces an overview of related work. In section 3, the coverage information is used to develop test inputs for a set of queries obtained from a real-life application. In section 4, we describe experimental and the evaluation of the coverage of SQL queries is performed. Finally, we conclude in section 5.

RELATED WORKS

Over the last fifteen years or so, much of the NLI community has focused on the use of statistical and machine learning techniques to solve a wide range of problems in parsing, machine translation, and more. Yet, classical problems such as building Natural Language Interfaces to Databases (NLIs) are far from solved.

Even though a great deal of research in software testing has been carried out in recent years, few studies have been specifically related to the testing of database applications, whether for test input selection criteria or test input adequacy criteria. An initial way of classifying the related work is in relation to the information used to meet the criteria: only the database, only the queries, and both of them. The selection of test inputs by means of considering the database schema and constraints though, not the application code is the approach taken by [4]. In order to automatically load the initial database, a set of valid and invalid data is generated from a database schema considering primary keys, null values and established ranges, but not referential integrity. Besides basing on database schema, select the test inputs using a set of non-deterministic rules like associations, correlations and patterns, and statistics of the current live data in the production database. The structure of the data and the query under test is considered in most studies on database testing. Present a theoretical approach using relational algebra and a notion of adequacy related to the concept of an Armstrong database. Queries, with select, project and join operations, expressed in relational algebra are represented as query graphs to be evaluated and a test database is generated for each given query after evaluating functional dependencies obtained from the database schema and the query. In this paper, apart from clauses included in parameters, grouping operations, aggregate functions and set quantifiers are considered, and selected test inputs for a query are evaluated according to a defined adequacy criterion.

These works are others of the most similar to this paper because they establish explicitly defined adequacy criteria, although SQL semantics are not taken into consideration. Also define a test adequacy criterion based on the coverage of all the SQL statements dynamically generated that an application can issue to a database. The approaches presented in are complementary to our approach and, in all cases; a common feature is that validation is performed using non-trivial systems or real-life applications, which inspires confidence in the capabilities of each method.

SQL QUERY MAPPING WITH QUESTION

Automatically mapping natural language into programming language semantics is a major and interesting challenge in the field of computational linguistics since it may have a direct impact on industrial and social worlds. For example, accessing a database requires machine-readable instructions that not everybody is supposed to know. Users should be able to pose a question in natural language without knowing either the underlying database schema or any complex structured machine language. The development of natural language interfaces over databases (NLIDBs), that translate the human intent into machine instructions used to answer questions, is indeed a classic problem that is becoming of greater importance in today's world.

Question. Then, to map new questions in the dataset of the available queries, (c) we rank the

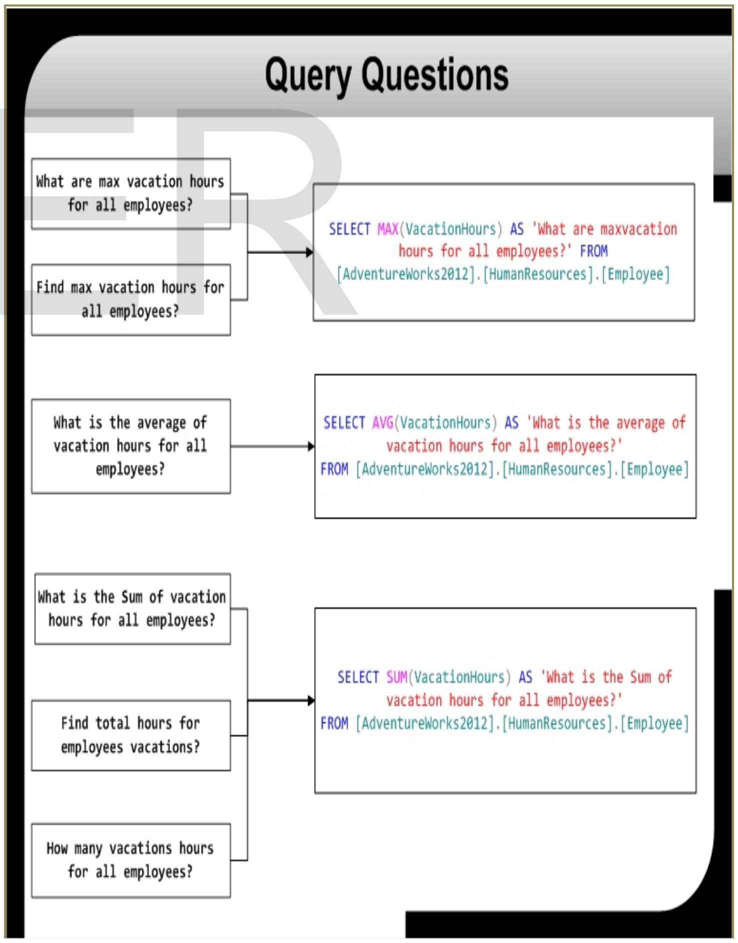
Studying the automatic mapping of questions into SQL queries is important for two main reasons: (a) it allows designing interesting applications based on databases (b) it offers the possibility to understand the role of syntax in deriving a shared semantics between a natural language and an artificial language.

We consider a dataset of natural language questions N and SQL queries S related to a specific domain/database and we automatically learn such mapping from the set of pairs P = N × S. More in detail, (a) we assume that pairs are annotated as correct when the SQL query answers to the question and incorrect otherwise (b) we train a classifier on the above pairs for selecting the correct queries for a

Latter by means of the question classifier score and by selecting the top one. In the following we provide the formal definition of our learning approach.
The first deals with the way in which the queries select and join information from different tables and the second with the way in which selected data is further processed. The most frequently used SQL statements in commercial applications are those that retrieve information (SELECT queries), that use a common set major characteristics, such as the database schema and the core clauses for projecting, joining, selecting and grouping data. However, developing a single statement may be a complicated task [5] and queries using GROUP BY, ORDER and HAVING clauses are considered especially difficult by programmers.
We provide a novel representation of the database schema by modeling all the tables in a single structure that enables the support for larger databases schemas. On the other hand, it is usual to use the same test database for a set of queries, as it reduces the cost of the test preparation and In this perspective, in real world domains, we may expect to have examples of questions and the associated queries which answer to such information need. Such pairs may have been collected when users and operators of the database worked together for the accomplishment of some tasks. In contrast, we cannot assume to have available pairs of incorrect examples, since (a) the operator tends to immediately provide the correct query and (b) both users and operators do not really understand the use of negative examples and the need to have unbiased distribution of them. Therefore, we need techniques to generate negative examples from an initial set of correct pairs see Fig.1.

SQL QUERY AND QUESTION TESTING

execution. Unlike many query-aware generation procedures, which generate one test database for each individual query of an application, our approach supports the automatic generation of a single test database for multiple queries within the application.
We present an approach for automatic populating test databases which employs a coverage criterion specifically tailored for SQL queries. Given a test database, a coverage rule holds if the execution of the corresponding SQL query against the test database produces at least one row as output. The coverage rules allow measuring the coverage of a test database against a set of queries or are used as a test input selection criterion.
We apply supervised techniques and, consequently, we need training data. More precisely, we need correct and incorrect pairs of questions and queries. Since in practical applications this is the most costly aspect, we should generate such learning set in a smart way.


Query Questions
What are max vacation hours for all employees?
Find max vacation hours for all employees?
What is the average of vacation hours for all employees?
What is the Sum of vacation hours for all employees?
Find total hours for employees vacations?
How many vacations hours for all employees?

```
SELECT MAX(VacationHours) AS 'What are maxvacation hours for all employees?' FROM [AdventureWorks2012].[HumanResources].[Employee]
```

```
SELECT AVG(VacationHours) AS 'What is the average of vacation hours for all employees?'
FROM [AdventureWorks2012].[HumanResources].[Employee]
```

```
SELECT SUM(VacationHours) AS 'What is the Sum of vacation hours for all employees?'
FROM [AdventureWorks2012].[HumanResources].[Employee]
```

Fig.1 Query Questions

Now, we will use Control Flow Graph (CFG) for illustrating coverage criteria and showing suggestions paths first use aggregate functions in attributes and define condition by using logical conditions or Group by or operators to filter data as see fig. 2.

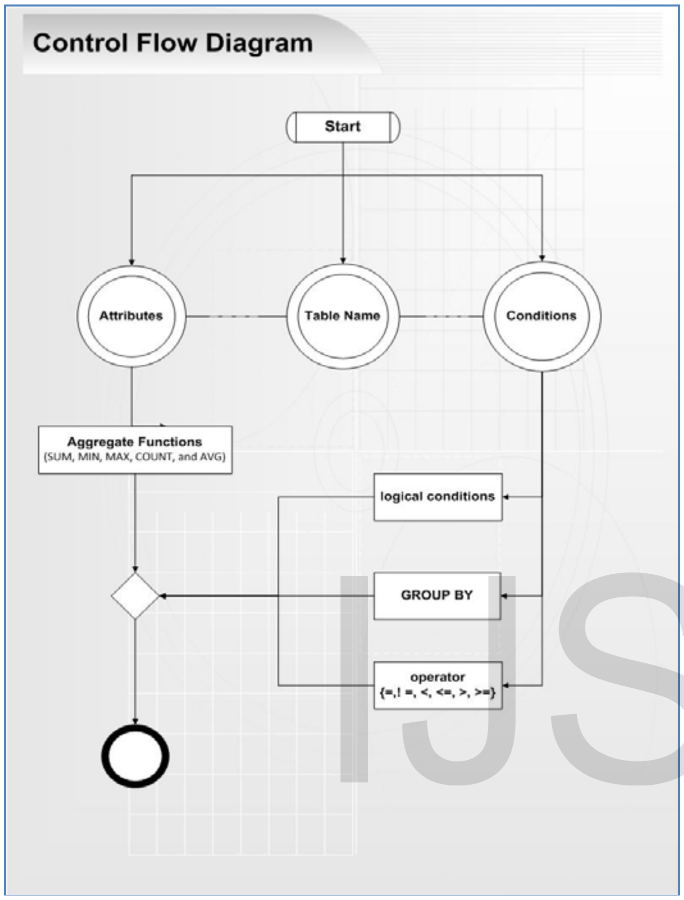


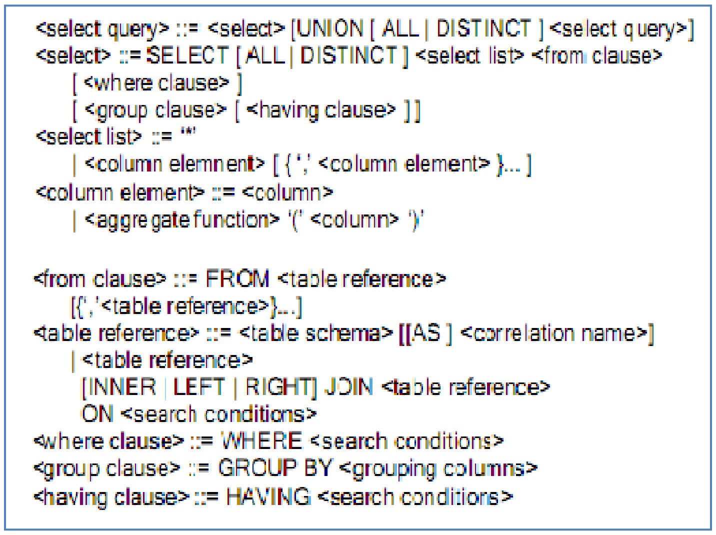
Fig. 2 SQL Query Control Flow Graph (CFG)

We will use control flow graph to illustrate SQL query coverage criteria Attributes and Conditions, we can use aggregate function (SUM, MIN, MAX, COUNT, and AVG) in query from user question and also map conditions using logical condition (AND,OR,NOT) or Group by or Operator with mapping from user question. We can use aggregate functions with query parameters without conditions or we can add one condition or more than one condition, we have different paths after mapping query with user question [6]. We can define paths by passing actual parameters from user question to actual parameters to SQL Query.

APPLY SQL QUERY COVERAGE CRITERIA

The goal of this paper is to define coverage criteria for assessing the adequacy of the test suite to exercise various situations that affect the data retrieved by an SQL query. The approach studies queries in an isolated way without considering the imperative code where they will be embedded and the tests can be used as prerequisites for embedding queries in the imperative code.

This paper improves where queries only had FROM and WHERE clauses and conditions were exclusively composed of attributes, constants or NULL. The present paper also considers parameters, GROUP BY and HAVING clauses, aggregate functions, ALL and DISTINCT quantifiers along with UNION operator. Moreover, it shows how to automate the calculation of the coverage and it analyzes different kinds of faults in queries classified in two categories: non SQL-specific (but typical faults in the conditions in imperative programs) and SQL-specific. The approach involves building one or more coverage nodes that are created on the basis of the structure of the query and the database schema. The subset considered in this paper is that represented in the following BNF grammar:



```
<select query> ::= <select> [UNION [ ALL | DISTINCT ] <select query>]
<select> ::= SELECT [ ALL | DISTINCT ] <select list> <from clause>
     [ <where clause> ]
     [ <group clause> [ <having clause> ] ]
<select list> ::= '*'
     | <column elemnent> [ { ',' <column element> }... ]
<column element> ::= <column>
     | <aggregate function> '(' <column> ')'

<from clause> ::= FROM <table reference>
     [{',' <table reference>}...]
<table reference> ::= <table schema> [[AS ] <correlation name>]
     | <table reference>
      [INNER | LEFT | RIGHT] JOIN <table reference>
      ON <search conditions>
<where clause> ::= WHERE <search conditions>
<group clause> ::= GROUP BY <grouping columns>
<having clause> ::= HAVING <search conditions>
```

Fig.3 Query Coverage Criteria

We will use different operators in search conditions to filter query data in <Search conditions> term is a logical predicate composed of logical conditions concatenated with AND , OR operators. A condition is an expression in the query in the form X R Z where X and Z are sets of values represented by the name of their column [7], aggregate functions, constants, parameters or NULL, R is an operator of {=,! =, <, <=, >, >=} and an aggregate function (count, sum, max, min or avg) transforms a set of scalars or a set of rows into a scalar.

CONDITION

We can use simple condition with one parameter using operator of {=,! =, <, <=, >, >=} or complex condition by adding more than one condition and using logical operators {AND, OR, NOT} between theses condition in where clauses and we can use GROUP BY or HAVING after conditions.

For example, what are departments that group name is Research and Development OR Manufacturing?

QUERY PARAMETERS

We will use parameters @1, @2 and @3 in different conditions to pass values to SQL Queries for testing such Q1 we will pass 3 different parameters values to get hire date that is great than @1 and less than @2 after get result we add logical operator AND to compare data with parameter @3 that all parameters with same data types. In Q3 we will use parameters with different data types @1 to pass value to contact type,@2 to pass value to contact name and so on for remaining queries.

AGGREGATE FUNCTIONS

The aggregate functions (SUM, MIN, MAX, COUNT, and AVG) perform simple calculations over all values that are included in each group. Additionally, SUM, COUNT and AVG can specify the optional set quantifier DISTINCT, which, if present, excludes the repeated values from the calculation. Two conditions that affect the calculation of the aggregate function are considered: (1) if some values are repeated, then only one value is taken into account if the DISTINCT set quantifier is present and (2) if a value is NULL, then it is not taken into account.

For example, what is the Sum of vacation hours for all employees?

GROUP BY

The GROUP BY clause indicates how to combine the selected rows and the HAVING clause performs a final filter based on other criteria (having-conditions).Let Q be a query with a GROUP BY clause composed of a list of grouping columns A1...Ac (each of them is either the name of a single attribute or an expression over the values of attributes). In this case the select-list is in the form A1...Ac, Fc+1...FN, where each Fi is an aggregate function expression over the values of attributes. According to SQL specification [SQL 1992], groups are partitioned "into the minimum number of groups such that for each grouping column of each group, no two values of that grouping column are distinct". As the semantics of the GROUP BY clause interprets null values in the grouping column as belonging to different groups, two conditions are considered, one in which grouping columns are not NULL and another in which they are.

EVALUATING CONDITION COVERAGE

We use a condition coverage CC for representing all possible combinations of the results of evaluation of the conditions of an SQL query. Each condition C, extracted from Q (which includes the conditions in the JOIN and/or WHERE clauses),

Given a SELECT query Q, its corresponding condition coverage CC (CS) is constructed by considering the ordered set of conditions CS=(C1,………..,Cs) of Q. After evaluating all condition coverage, some of the values are covered and some others remain uncovered. The theoretical condition coverage is calculated as the percentage of covered c-values. As this measure does not consider impossible c-values, it is necessary to define another that takes them into account, the schema condition coverage (c-coverage):

$$\text{C-coverage} = \frac{sum(covered\ c\text{-}values)}{sum(total\ c\text{-}values) - sum(impossible\ c\text{-}values)} \times 100$$

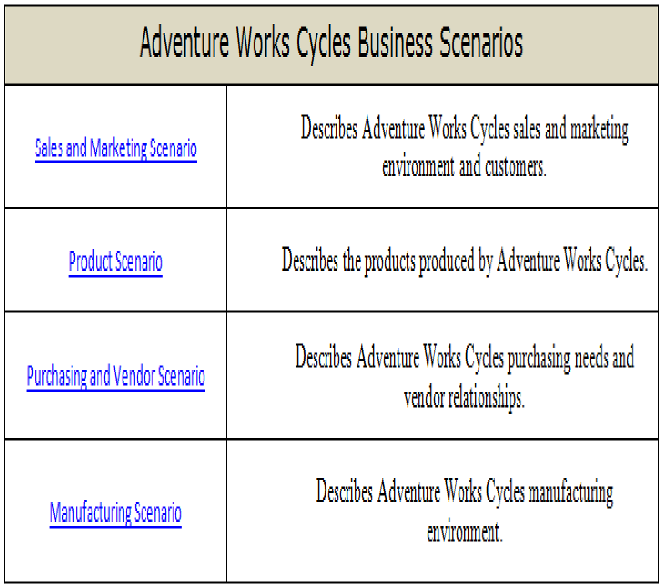
Fig.3 Condition Coverage

A c-value is labeled as 'N' if it is not covered, 'Y' if it is covered, 'I' (impossible) if it cannot be covered due to some known restriction imposed by the database schema and 'U' (unreachable) if it cannot be covered because of characteristics or constraints that do not depend on the database schema, such as the condition, their operands, constants or parameters. Note that if database schema is modified

affecting any attribute of the condition, the impossible c-values could change with the new constraints. Initially, each c-value is labeled 'N' meaning that the c-value has not been covered yet. Moreover, the c-values impossible to cover because of the database schema are automatically labeled as 'I'.

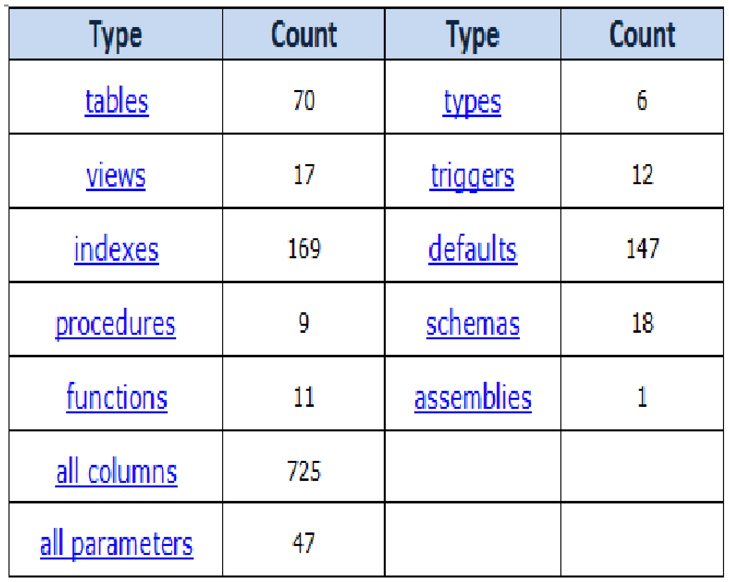
EXPERIMENTAL EVALUATION

Adventure Works Cycles, the fictitious company on which the AdventureWorks sample databases are based, is a large, multinational manufacturing company. The company manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in Bothell, Washington with 290 employees, several regional sales teams are located throughout their market base. In 2000, Adventure Works Cycles bought a small manufacturing plant; Import adores Neptune, located in Mexico. Importadores Neptuno manufactures several critical subcomponents for the Adventure Works Cycles product line. These subcomponents are shipped to the Bothell location for final product assembly. In 2001, Import adores Neptune became the sole manufacturer and distributor of the touring bicycle product group. Coming off a successful fiscal year, Adventure Works Cycles is looking to broaden its market share by targeting their sales to their best customers, extending their product availability through an external Web site, and reducing their cost of sales through lower production costs.



Adventure Works Cycles Business Scenarios	
Sales and Marketing Scenario	Describes Adventure Works Cycles sales and marketing environment and customers.
Product Scenario	Describes the products produced by Adventure Works Cycles.
Purchasing and Vendor Scenario	Describes Adventure Works Cycles purchasing needs and vendor relationships.
Manufacturing Scenario	Describes Adventure Works Cycles manufacturing environment.

Fig.4 Adventure Works Cycles

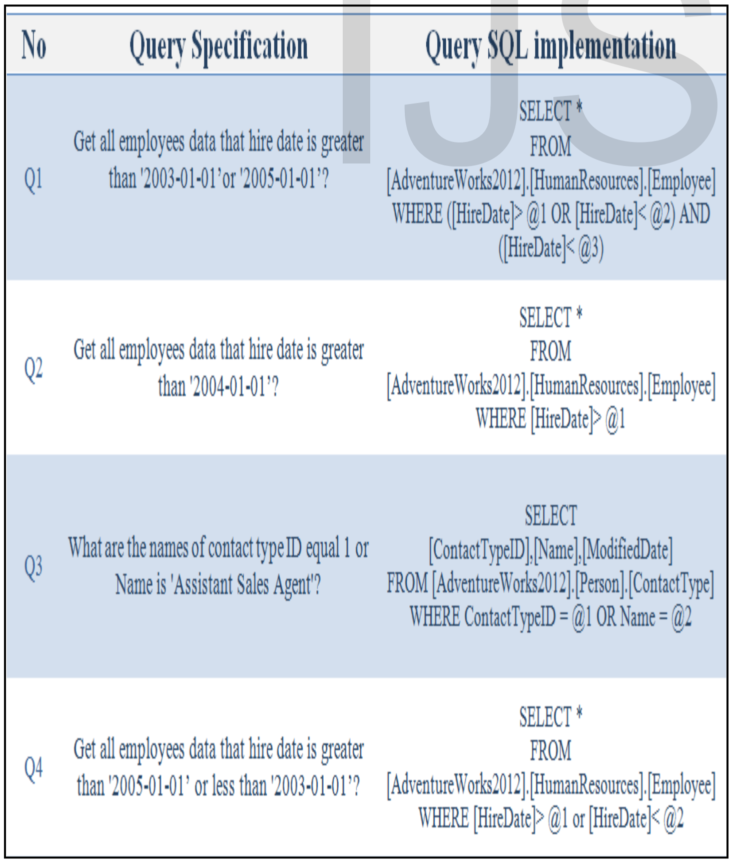


Type	Count	Type	Count
tables	70	types	6
views	17	triggers	12
indexes	169	defaults	147
procedures	9	schemas	18
functions	11	assemblies	1
all columns	725		
all parameters	47		

Fig.5 Adventure Works Objects

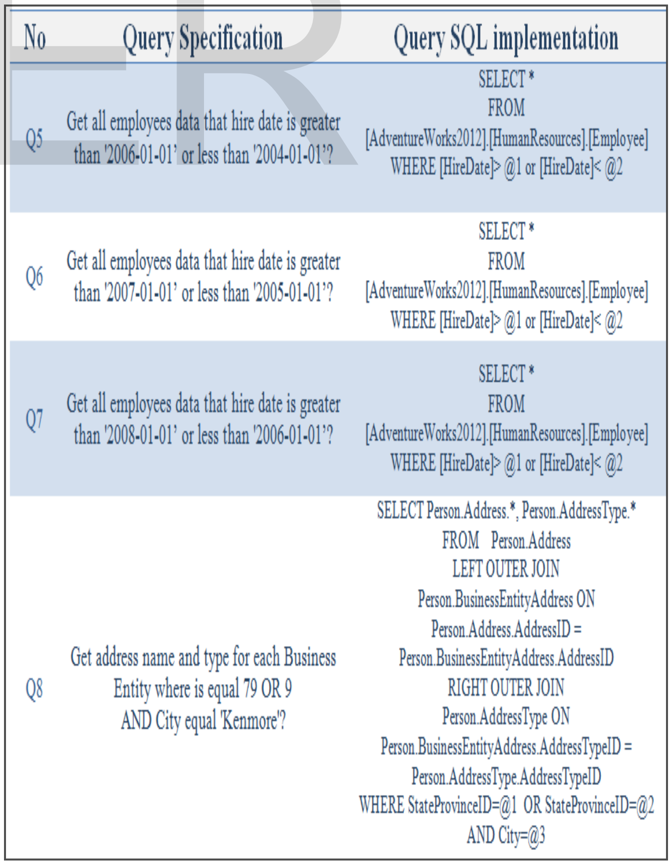
You can get more details about AdventureWorks Sample OLTP Database [11]:

DATABASE OBJECTS



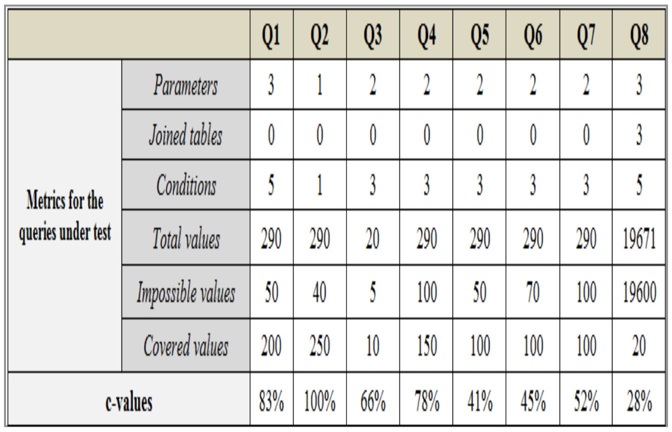
No	Query Specification	Query SQL implementation
Q1	Get all employees data that hire date is greater than '2003-01-01'or '2005-01-01'?	SELECT * FROM [AdventureWorks2012].[HumanResources].[Employee] WHERE ([HireDate]> @1 OR [HireDate]< @2) AND ([HireDate]< @3)
Q2	Get all employees data that hire date is greater than '2004-01-01'?	SELECT * FROM [AdventureWorks2012].[HumanResources].[Employee] WHERE [HireDate]> @1
Q3	What are the names of contact type ID equal 1 or Name is 'Assistant Sales Agent'?	SELECT [ContactTypeID],[Name],[ModifiedDate] FROM [AdventureWorks2012].[Person].[ContactType] WHERE ContactTypeID = @1 OR Name = @2
Q4	Get all employees data that hire date is greater than '2005-01-01' or less than '2003-01-01'?	SELECT * FROM [AdventureWorks2012].[HumanResources].[Employee] WHERE [HireDate]> @1 or [HireDate]< @2

We will use standard Microsoft Adventure Works database for testing our application, our experimental evaluation as follow. We will have 8 questions with 8 SQL query examples as follow:



No	Query Specification	Query SQL implementation
Q5	Get all employees data that hire date is greater than '2006-01-01' or less than '2004-01-01'?	SELECT * FROM [AdventureWorks2012].[HumanResources].[Employee] WHERE [HireDate]> @1 or [HireDate]< @2
Q6	Get all employees data that hire date is greater than '2007-01-01' or less than '2005-01-01'?	SELECT * FROM [AdventureWorks2012].[HumanResources].[Employee] WHERE [HireDate]> @1 or [HireDate]< @2
Q7	Get all employees data that hire date is greater than '2008-01-01' or less than '2006-01-01'?	SELECT * FROM [AdventureWorks2012].[HumanResources].[Employee] WHERE [HireDate]> @1 or [HireDate]< @2
Q8	Get address name and type for each Business Entity where is equal 79 OR 9 AND City equal 'Kenmore'?	SELECT Person.Address.*, Person.AddressType.* FROM Person.Address LEFT OUTER JOIN Person.BusinessEntityAddress ON Person.Address.AddressID = Person.BusinessEntityAddress.AddressID RIGHT OUTER JOIN Person.AddressType ON Person.BusinessEntityAddress.AddressTypeID = Person.AddressType.AddressTypeID WHERE StateProvinceID=@1 OR StateProvinceID=@2 AND City=@3

Summarizes all the results obtained after using coverage:



		Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Metrics for the queries under test	Parameters	3	1	2	2	2	2	2	3
	Joined tables	0	0	0	0	0	0	0	3
	Conditions	5	1	3	3	3	3	3	5
	Total values	290	290	20	290	290	290	290	19671
	Impossible values	50	40	5	100	50	70	100	19600
	Covered values	200	250	10	150	100	100	100	20
c-values		83%	100%	66%	78%	41%	45%	52%	28%

Fig.6 Query Matrix

We apply test case on AdventureWorks2012 database different schemas and tables .first schema [HumanResources] with table [Employee] that contains 290 records with different queries and conditions apply on query
(Q1, Q2, Q4, Q5, Q6, Q7), second schema [Person] with table [ContactType] that contains 20 record with different queries and conditions apply on query (Q3) finally using [Person] schema with different tables [Address], [BusinessEntityAddress] and [AddressType] that contains 19671 records with different queries and conditions apply on query (Q8) using join operators to select data from more than one table then we Covered values as examples to apply test for system.

CONCLUSION

Adequacy criteria provide an objective measurement of test quality. Although these criteria are a major research issue in software testing, little work has been specifically targeted towards the testing of database-driven applications. In this paper, two structural coverage criteria are provided for evaluating the adequacy of SQL queries that retrieve information from the database. We evaluate the approach on an industrial case study including a number of queries and a schema with a large number of tables and columns by generating set of Coverage Rules for each condition in where clause. we will use control flow diagram to illustrate SQL query coverage criteria Attributes and Conditions, we can use aggregate function (SUM, MIN, MAX, COUNT, and AVG) in query from user question and also map conditions using logical operators (AND,OR,NOT) or Group by with mapping from user question. We can use aggregate functions with query parameters without conditions or we can add one condition or more than one condition.

This paper improves queries that only had FROM, WHERE clauses and conditions were exclusively composed of attributes, constants or NULL. Also we consider parameters, GROUP BY and HAVING clauses, aggregate functions, ALL and DISTINCT quantifiers along with UNION operator. Finally we apply SQL Query coverage criteria using Conditions. We can use simple condition with one parameter using operator of {=,! =, <, <=, >, >=} or complex condition by adding more than one condition and using logical operators {AND, OR, NOT} between theses condition in where clauses and we can use GROUP BY or HAVING after conditions. Aggregate Functions The aggregate functions (SUM, MIN, MAX, COUNT, and AVG) GROUP BY clause indicates how to combine the selected rows and the HAVING clause performs a final filter based on other criteria.

REFERENCES

[1] Zahraa E. Mohamed, Mona Gharib, Mohamed Reda:" Intelligent Multidimensional Database Interface "International Journal of Scientific & Engineering Research, Volume 4 |(11), 2013.

[2] I. Androutso poulos, G. D. Ritchie, and P. Tarnish." Natural Language Interfaces to Databases –
An Introduction. In Natural Language Engineering", volume 1, part 1, pages 29–81, 1995.

[3] Mª José Suarez-Cabal. And Javier Tuya. "Structural Coverage Criteria for Testing SQL Queries" Journal of Universal Computer Science, vol. 15(3), 2009.

[4] Binnig, C., Kossmann, D., Lo, E. 2007. "Reverse query processing". In Proceedings of the 23rd International Conference on Data Engineering. IEEE Computer Society, Washington, DC.

[5] Tuya, J., Suarez-Cabal, M.J., de la Riva, C. 2010. "Full predicate coverage for testing SQL database queries". Softw. Test. Verify. Real, in press.

[6] Brass, S., Goldberg, C.: 2005 "Semantic Errors in SQL Queries: A Quite Complete List (Extended version)"; Journal of Systems and Software 79, 5 (), 630-644.

[7] Tuya, J., Suarez-Cabal, M.J., De la Riva, C., 2006, "SQL Mutation: a Tool to Generate Mutants of SQL Database Queries"; 2nd Workshop on Mutation Analysis.

[8] Bruno, N., Chaudhuri, S. 2005. Flexible database generators. In Proceedings of the 31st International Conference on Very Large Data Bases. VLDB Endowment,

[9] Tuya, J., Suarez-Cabal, M.J., De la Riva, C. 2007; "Mutating Database Queries"; Information and Software Technology, 49, 4 .

[10] Adventure Works Cycles Business Scenarios
http://technet.microsoft.com/en-us/library/ms124825(v=sql.100).aspx

[11]Adventure Works databases ummary
http://www.dbdesc.com/output_samples/htmlbrowse_AdventureWorks.html