**Literature Review**

**Chays, 1999** In this paper the author have described the tool called as AGENDA, (A test GENerator for Database Applications), which facilitate the testing of relational database applications. In testing such application, the state of the database before and after the user's operation plays an important role, along with user's input and systems output. A framework for testing database application is introduced. A complete tool set, based on this framework has been prototyped. The components of this system are a parsing tool that gather relevant information from the database schema and application, a tool that populates the database with meaningful data that satisfy database constrains, a tool that generates test cases for the application, a tool that checks the resulting database state after operations are performed on a database application and a tool that assists the tester in checking the database application’s output.

**Halfond, Willmor, 2002** In previous paper, the tool described by David Chays was designed to aid in testing database applications. In addition to the tools that agenda currently has, three additional tools were made to enhance testing and feedback. They are the log analyzer, attribute analyzer, and query coverage. The log analyzer finds relevant entries in log file produced by DBMS, lexically analyze them using a grammar written in Javacc, and stores some of the data in a database table. when the log entry represent an executed SQL statement, this statement is recorded. The attribute analyzer parses SQL statement. A SQL grammar for Javacc was modified, adding code to determine which attributes are read and written in each SQL statement. A new test coverage criterion query coverage is defined. query coverage check whether queries that the tester thinks should be executed actually are executed. Similar to log analyzer javacc was used to implement this. it is implemented by pattern matching executed queries identified by the tester.

**Suárez, 2003** In the paper written by María José Suárez, the author have explained the Alloy which is a first-order declarative language based on sets and relations. An Alloy model consists in a set of signatures (sig) which enables the definition of elements and a set of relations (fields) between them, and a set of constraints that restricts the space of the model. Constraints in Alloy can be of two types. A fact (fact) is a constraint that always holds. Predicates are constraints
formulas whose satisfiability needs to be checked. The Alloy analyzer translates the specification into propositional formulas and generates a set of finite scope instances that satisfy the constraints. A previous approach in test database generation with Alloy is based on modeling each table of the schema as a signature consisting of a relation over the domains of each attribute, that is, each table is modeled with a relation defined as the Cartesian product of its column domains (for a table with n columns, a n-arity relation). Database constraints are specified as fact constraints over the signature of each table. Although this representation is consistent with the relational model, it has the main drawback of its scalability. Because each table of the database is represented as the Cartesian product of its column domains and the analyzer enumerates all the solutions (test databases) that satisfy the specification, it could be infeasible to process large tables as the state space to analyze could be much larger. In our preliminary experiments, the Alloy analyzer ran out of memory in databases with tables with more than five columns (5-arity relations).

Urpi, 2004 In the said paper, the author described the concept of schema validation, for which he had used the SVT which is known as schema validation tool. It allows a database designer to perform several test to check desirable properties of database schemas defined in SQL Server. Among them we have: schema satisfiability, liveliness, integrity constraint redundancy, reach ability, etc. SVT is able to check whether a given property is satisfied or not. SVT accepts schemas defined by means of a subset of the SQL language provided by SQL server. It accepts the definition of: primary key, foreign key, Boolean check constraints, views, negation, sub selects union, data types.

The current SVT validate the schema by means of testing the desirable properties like state satisfiability, Liveliness, Integrity constrains redundancy, reach ability and query containment, of the database schema.

Halfond , 2005 In the paper written by William, he has mentioned an approach for measuring the coverage of an SQL query based on the coverage concept whereby the conditions take into account the true and false values during the explorations of their different combinations. Given the variety of SQL statements that can be found in an application, we have restricted these to a subset of SELECT queries specified in SQL3, according to the grammar in BNF notation shown,
in order to first achieve testing with simple SQL queries, to subsequently extend the analysis to other, more complex queries.

**Duhan, Torp, 2006** In this paper the authors have used the base of coverage metric using Test agent that coordinates the conversion agent and test suite agent. The test agent activates each agents depending on the messages received from the agents and activates the other agents accordingly. During database testing, the test agent initiates the conversion agent first and checks whether the table conversion (Database table to XML file) has been completed first. After the completion of the work by the conversion agent, it activates the test suite agent. The test suite agent executes the test cases for the database from repository. After the completion of the test case execution, the test suite agent response to the test agent through a message. The test agent initiates the coverage metric agent which measures the effectiveness of the test process through various metrics that enhances the quality of the test process.

**Willmor, 2006** In this paper the author have presented a novel test input generation algorithm that tracks not only the program state but also the environment (the state of the database). While his early results are encouraging, he have identified some limitations of the presented approach. First, their implementation assumes a view of the world with two participants: Java programs and the database. In reality, most enterprise applications are built in several different layers, including JavaScript code, browser forms, and a server such as tomcat that mediates data. While conceptually the algorithm remains the same, we admit that scaling our implementation to a real enterprise system is a significant engineering effort. Second, symbolic execution based test generation is ultimately limited by the expressibility of the constraint language and the capacity of the constraint solver. The author believe that constraint solver presents a compromise between fast constraint solving and the ability to capture many constraints of practical interest. Despite these limitations of our current implementation, we believe that context-aware co colic execution presents a powerful tool for automatic test generation and validation of database-driven applications.
**Willmore and Suzanne, 2006** These two authors have suggested structural adequacy criteria for testing database applications, which is based on intentional specification of database state and intentional descriptions of the behavior of database operations. The criteria presented by them focused on the structural and data oriented elements of database systems. The structural elements include branches, loops and procedure calls. Data oriented elements are the points at which data is defined and used in the program. The fundamental issues they have considered in database testing are, whether the application behave specified, the fault like attempting to access a database entity that does not exist, operations attempting to violate the database constrains and transaction being aborted or committed incorrectly.

The structural test adequacy criteria are based on static analysis of the model according to a specific criterion. These structural criteria can be seen as analogous to traditional control flow based criteria but with the additional characteristics of database system incorporated. In the structural criteria they have used the concept of complete path that starts at the graph's entry node and ends at an exit node. In this method it can be observed that execution of a test case will result in execution of a particular complete path. It is to be noted that as the initial state of the database is described using the intentional constraints, multiple execution of test case may result in different paths due to changes in the database state. Therefore a test is adequate in terms of the state in which it was executed.

**Haftmann, 2007** In the mentioned paper the system structure is divided into subgroups that can be independently certified. A set of tests is enumerated for each subgroup. When a means of testing is provided for all constructs of the conceptual model, the testing is said to be exhaustive. A matrix form can be used to demonstrate the completeness of the testing.

The approach is targeted to generate test cases for testing database applications. Making use of the conceptual model of the application database, test cases can be designed to test all constructs of the ER model and the constraints of the databases. With the input of the user defined functions, test cases can be generated to test the database functions supported by each entity. Different testers may produce different sets of tests cases in order to examine all constructs of the conceptual model. This may significantly influence the effectiveness of testing. Like Black Box testing, this approach may miss those faults arisen from improper execution sequences.
In this paper the author have worked on the concept suggested by researchers Mr. Chan and Cheung, which is known as WHODATE, which stands for WHite bOx Database Application TEsting, to generate additional test cases from embedded SQL statements. In this approach, the embedded SQL statements are transformed into General Purpose Language(GPL) statements. Conventional white box techniques are then applied to both the transformed statements and other statements written in the host language. One of the goals of this approach is to include the semantics of the SQL statements in test case generation that is to test whether the semantics of the SQL statement can be treated as black boxes and be tested separately. But this approach may not be possible because no functional specification is defined for the SQL statements.

In this paper the author have given detail information about testing of Database application, since the testing of database application is quite different as compared to other applications. According to the author the Database applications are generally be classified into two categories. The first category consists of application that are solely built in Data Manipulation Language (DML) and the language supported by the DBMS. In these applications, queries and transactions are specified in DML and user interfaces are defined in the DBMS language. The other category involves applications that are built in both DML and general purpose programming language such as C, C++, JAVA, Visual Basic Etc. Statements of Structured Query Language (SQL), a powerful declarative query language, are embedded in programs written in any general purpose language which is referred to as the host language. In these applications, queries and modification of data are written as the embedded SQL statements.

These authors have suggested the dataflow adequacy criteria for testing database application that is associated with the entities in a relational database. In addition to it they have developed a representation of a database driven application called a database interaction control flow graph in order to calculate the test adequacy matrices. The data flow based adequacy metric are particularly used to capture a program’s interaction with a database at multiple levels of granularity and to ensure the existence of test suits that can detect violation of database validity and completeness.
**Chan, 2008** In the paper by M. Y. Chan, he had described the tool called as RAGS(random generation). Since deterministic Testing of SQL database system is human intensive and cannot adequately covered the SQL input domain. A system RAGs was built to explore Automated testing. RAGS is currently used by Microsoft SQL server testing group.

RAGS  is used to drive one SQL System for observable errors such as lost connection, compiler error, execution errors and system crashes. The output of successful select statement can be saved for regression testing. If SQL Select executes without error, there is no easy method to validate the return values by observing only the values, the query and the database state.

**Farre, 2009** In this paper the author have used traditional black box testing technique for testing the database. According to the author the advantage of black box testing method is that test cases can be generated independent of the programs at an earlier stage of the software development cycle. Programmer can have the test case in mind when they develop an application. Thus the resultant application tends to satisfy most of the generated test cases. Relatively few efforts are required to test and debug the applications. Another advantage of using black box is that the cost generating test cases are rather low as compared to the cost associated with white box technique. Thus, the development cost can be reduced. Besides, many black box testing have already been well designed. Tester can select the most suitable one for testing their database application.

On the other hand, without examining the program, we do not know how much of an application is being tested. Practically, functional specifications are generally specified in natural languages. But using the natural language for writing speciation can causes problems including redundancy, inconsistency and incompleteness. In addition black box testing are insensitive to some kind of faults like, suppose a function of database application (say A) is said to be dependent on the functions (say B & C) when the set of relations the former accesses overlaps with that accessed by the latter. The outcome of the function A may be affected by the functions B & C as the latter may change the internal state of the database. The function may fail after some sequence of execution of other functions. As such, it is necessary to like for techniques beyond black box testing.

**Haller, 2009** As in last review we have seen that the author had used the black box approach, the author of this paper had given emphasis on the traditional white box testing, which is a techniques like statement testing, branch testing, condition coverage and path testing can be
employed to test some portion or the whole database application to attain a more complete testing of the application. White box testing lets tester examine the decode in detail and make sure that at least a certain degree of test coverage such as execution of every statement has been achieved.

In fact traditional white box techniques have their own limitations in testing database applications. Most importantly, they have not explicitly considered the SQL statements embedded in application programs. The SQL statements are treated as black boxes. Usually, only a few test cases are generated by white box techniques to test the embedded SQL statements. No test cases are generated intentionally to include the semantics of the SQL statements, which reflect the change of the internal database states. It may be possible that the traditional white box testing may miss the type of faults related to the internal database change.

Mahmood and He, 2009 In this paper the researchers have developed a formula for hypothetical database testing by using test suites requirements. When tester testing by using test suites requirements, executes test suites requirements one by one, the formula automatically decides, whether the test suites say ti, where i=1, 2, 3, ..., k is executed hypothetically or traditionally. In other words the computer aided formula (or Mathematical function) which tells us when to update-in place and when to rollback according to the environment of input requirements of test suites. For designing the formula below using the concept of “Functional Analysis” a well known branch of Mathematics, it is claimed that the said formula of this type has not been used in current state of art and may be regarded a novelty in the field of software testing. The formula is: f(n,mi,ri+1)=2ri+1–(n+mi)

Hierons, and Gould, 2009 In the written paper the author have used the concept of partitioning the large domain in to sub domain. Partition analysis we divide the input domain to form sub domains on which the system’s behavior should be uniform. Boundary value analysis produces test inputs near each sub domain’s boundaries to find failures caused by the boundaries being incorrectly implemented. However, boundary value analysis can be adversely affected by coincidental correctness the system produces the expected output for the wrong reason. This paper shows how boundary value analysis can be adapted in order to reduce the opportunity for
coincidental correctness. The main contribution is to automate test data generation in which one cannot rely on the expertise of a tester.