A Comprehensive Approach to Anomaly Detection in Relational Databases

Adrian Spalka[1] and Jan Lehnhardt[2]

[1] Dept of Computer Science III, University of Bonn
Römerstr. 164, 53117 Bonn, Germany
adrian@iai.uni-bonn.de
[2] NOVOTERGUM AG
Im Park 20, 50996 Köln, Germany
j.lehnhardt@novotergum.ag

Abstract. Anomaly detection systems assume that a certain deviation from the regular behaviour of a system can be an indicator for a security violation. They proved their usefulness to networks and operating systems for a long time, but are much less prominent in the field of databases. Relational databases operate on attributes within relations, ie, on data with a very uniform structure, which makes them a prime target for anomaly detection systems. This work presents such a system for the database extension and the user interaction with a DBMS; it also proposes a misuse detection system for the database scheme. In a comprehensive investigation we compare two approaches to deal with the database extension, one based on reference values and one based on Δ-relations, and show that already standard statistical functions yield good detection results. We then apply our methods to the user interaction, which is split into user input and DBMS behaviour. All methods have been implemented in a semi-automatic anomaly detection tool for the MS SQL Server 2000.

Keywords: Database security, anomaly detection, misuse detection, relational databases.

1 Introduction

Today's relational databases and database management systems (DBMS) offer a variety of protection mechanisms. As a prerequisite, users must pass the identification and authentication to obtain access to a DBMS. A user's powers at the DBMS-level, eg, the ability to perform data definition operations, to backup databases or to act as an administrator, are often constrained with privileges. Also at the DBMS-level is an access control system, which decides on a user's access to individual databases within the DBMS. Security at the database-level relies mostly on the mechanisms provided by SQL, the standard language for relational databases. Database developers are assumed to enforce confidentiality through authorisation, ie a restriction of access to relations and views with

S. Jajodia and D. Wijesekera (Eds.): Data and Applications Security 2005, LNCS 3654, pp. 207–221, 2005.

grant/revoke statements. The preservation of availability resorts to technical means, such as backups and execution time limits for queries.

Let us now turn our attention to the two concepts of integrity and accuracy. The (present) extension of a database, ie the data in its relations, should be an accurate image of the present state of the corresponding real-world section. A database supports accuracy by means of integrity, which is defined by a set of constraints. In SQL, data types and primary-key constraints are examples of declarative integrity constraints and triggers are used to specify operational integrity constraints.

From a database's viewpoint, integrity constraints separate the definitely inaccurate data sets from the possibly accurate data sets, which are admitted as extensions. In practice, the set of possible extensions of a database is very large. And thus, though we know that the extension is always possibly accurate, all its data can still be wrong. Well, every reasonably large database has a small fraction of inaccurate data – a well designed business process can cope with it. However, the larger this fraction grows the larger is its negative impact.

Anomaly detection is a technique that generates hints of probably wrong data and harmful operations. In a first step an anomaly detector examines the regular state and behaviour of a system and computes from them a set of reference data, which captures their characteristic properties. Then, the same computations are applied to the system in operation and the current set is compared with the reference set. Whenever the difference exceeds a specified threshold, the anomaly detector reports an anomaly, viz an unusual deviation.

Networks benefit from this idea for a long time. Intrusion detection systems (IDS) are the most popular type of anomaly detectors (there are countless references). But also operating systems are a prominent target for anomaly detectors (cf eg [2], [4] and [8]).

Anomaly detection works best, ie produces the fewest wrong hints and alarms, on systems with clear patterns of regularity. The identification or extraction of these patterns is the most difficult task in the design of an anomaly detection system (ADS) for networks and operating systems – with well designed relational databases many of them come for free.

Our ADS is based on the following facts and assumptions:

- An attribute in a relation has a simple data type. This guarantees a basic uniformity of its values, which can be exploited by specific functions.
- The extension of a database changes in a smooth way.
- A user executes syntactically related commands, which place a specific load on the DBMS.
- Some elements in a database scheme, eg integrity constraints and indexes, are particularly important for the security of the database.

Speaking in terms of operating systems, an ADS can operate in real-time or in batch-mode. A real-time operation depends mainly on three conditions:

- the ADS can monitor or collect system data at short time intervals
- the ADS can evaluate this data with a low overhead
- the ADS can make a decision on a possible anomaly based on this data

Our assumptions about a database do not meet these conditions. Long running transactions but also single SQL-statements, which affect a large number of tuples, obstruct a timely collection of data. The search for an anomaly can involve large parts of a database and, thus, may require a lot of time. And, lastly, many deviations from the assumed smoothness will only be caused by substantial amounts of data. We therefore design our ADS as a batch-mode system, which is executed at a time of low activity in the database, eg during night. The distance between two runs of the ADS is dictated by the environment in which the database is used. In general, once a day or once a week should be a good choice. We regard our ADS as an augmentation to preventive security systems for the ADS only checks if anomalies or misuse have occurred[1].

Our ADS is composed of three components:

- An anomaly detector for the database extension
- An anomaly detector for the user interaction
- A misuse detector for the database scheme

The main emphasis in the construction of the ADS is placed on the first component. Here we offer two approaches. The first one is based on the comparison of reference values, which are obtained with a combination of fairly basic statistical functions on the elements of single attributes. It yields surprisingly good results, so that we dropped – or at least postponed – the initial intention of applying data mining techniques to the extension. Although its time and space requirements are very modest, the detection process works best on databases in which deletions or updates of a large number of tuples occur only seldom. The second approach uses Δ-relations that record the history of changes of the values of the monitored attributes between two runs of the ADS. On the positive, it can be tuned to precisely detect every kind of misuse; on the negative, it can require considerable additional space.

The above-developed analysis techniques are then used in the second component for anomaly detection in the user interaction. It computes reference values from two sources: the user input, ie the SQL-command strings, and the resulting behaviour of the DBMS. This component can detect operations that are admitted by the authorisation controls of the DB/DBMS and yet violate a company's security policy, eg due to the abuse of rights by the legitimate user or masquerading by an intruder.

The third component should have become an ADS for the database scheme, but (according to [1]) it turned out to be a misuse detection system (MDS). Here we are interested in commands, in particular SQL data definition language (DDL) statements, that severely impair the security, including the availability, of a database. We store a list of possibly dangerous commands in a library of signatures and compare the current command to it. Each entry in the library is associated with a critical element in the schema and a possibly damaging

[1] We would like to mention that our ADS, like every threshold-based protection system, will raise a false alarm in case of an unusual normal change, and will fail to detect an attack that complies with our normality-rules.

command on it. This allows us, for example, to relate a performance degradation with a dropped index.

The first two components of the outlined ADS are implemented for the MS SQL Server 2000. Presented with a graphical user interface, a user can select relations or attributes that should be monitored. The ADS then generates the appropriate relations and monitoring routines. The derived reference values offer a guide to the user for the initial setting of alarm thresholds. At present, an email is sent to the administrator if the ADS detects a violation. But the ADS also provides a full graphical evaluation of its run or a history of runs.

The subsequent section comments on previous and related work in this area. The ADS component for database extension is presented in detail in section 3 and that for user interaction in section 4. Section 5 describes the approach to misuse detection for the database scheme. Section 6 illustrates the operation of our ADS on an example database. Lastly, a summary concludes this work.

2 Previous and Related Work

There are numerous works on IDS for operating systems and networks, but not on databases. And there are numerous works on database security, but not on anomaly detection. Hence, we are confronted with a fairly small group of works related to our approach.

The DEMIDS system presented in [3] uses anomaly detection methods for the detection of misuse. It focuses on the misuse of privileges. At the core are frequent item-sets, which are computed in the training phase of DEMIDS for each user. These sets comprise relations, attributes and values which a user most often uses in his SQL-commands. The authors develop a distance measure between such a set and a command. In the real-time monitoring stage DEMIDS uses this measure to compute the distance between a user's frequent item-set and his actual query. If a threshold is exceeded, the system raises an alarm.

Our work is influenced by some ideas of the DAS system described in [5]. In the training phase DAS computes data related to a database's extension. With a data centric-view, the authors concentrate on numerical data types. They use the min, max, avg and stddev SQL-functions during the monitoring phase to detect unusual changes in the extension. Our work extends this approach to derive a variant of the frequent item-sets.

We would also like to mention DIDAFIT, a system introduced by [6]. It deals with SQL injection attacks, in particular on web-applications, which construct SQL-commands from parameters supplied by the user. DIDAFIT can be applied to exisiting applications, which are prone to this type of attacks, without re-programming the input validation. It modifies the semantics of an SQL-command with random data, derives for a user a general form of his commands and checks the difference between this form and the current SQL-command.

Lastly, [7] consider temporal objects in databases that register sensor data and present a method for checking for anomalies in the registration intervals.

3 Anomaly Detection in the Database Extension

This section describes two approaches to the detection of anomalies in the database extension.

3.1 Anomaly Detection Based on Reference Values

The anomaly detection based on reference values uses the following method. In the first step the user specifies the attributes, which should be monitored for anomalies. Then our ADS computes for the data of each attribute a set of reference values. The number and type of the values depend on the data type of the attribute. Now the user can specify thresholds for each reference value. [2] When the ADS is executed again, eg a day later, it performs the same computations on the data in the database and compares the current values with the previously computed reference values. If the difference exceeds the threshold, the ADS raises an alert; otherwise we assume that the database has evolved in a regular fashion and the current values replace the old reference values.

We now identify for each data type the corresponding parameters that can be used to capture the behaviour of that data. The MS SQL Server 2000 supports the following six groups of data types:

- bit
- integer, floating point and money
- ascii strings
- unicode and binary strings
- date and time
- unique id

We exclude several data types from our consideration. The type timestamp/rowversion represents system-generated global identifiers and sql_variant is a generic data type, which expose no useful regularity; the types cursor and table are not used in relations (only in stored procedures).

We now describe all reference values; a suitable subset of these values is associated with each group of data types, which is summarised in a table at the end of this section.

- OC: Overall Count.
  It represents the number of tuples in a relation. It is the only value associated with relations; all other values are associated with attributes. Let OC_1 denote the old value and OC_2 the current value. Then OC raises an alarm if the absolute change in the number of tuples, $OC_1 - OC_2$, or the relative change, OC_1/OC_2, exceeds the bounds defined by the threshold.
- NNC: Non-NULL-Count.
  It holds the number of non NULL values in the extension of an attribute. Its alarm conditions are analogous to those of OC.

[2] We later address the problem of finding the *right* thresholds.

- NNR: Non-NULL-Ratio.
  It is defined as NNC/OC. It detects insertions of a large number of null values. Its alarm conditions are analogous to those of OC.
- MIN, MAX, AVG, $STDEV$ and $RANGE$.
  These values are the results of the SQL functions minimum, maximum, average and standard deviation, and $RANGE = MAX - MIN$ applied to the extension of an attribute. Their alarm conditions are analogous to those of OC. The thresholds for MIN, MAX and $RANGE$ are often set to zero.
- RC_i, $i = 1, \ldots, 6$: Range Counters.
  The range counters monitor the distribution of the number of values of an attribute in the following six ranges:
  - RC_1: number of values below MIN
  - RC_2: number of values between MIN and $AVG - STDEV$
  - RC_3: number of values between $AVG - STDEV$ and AVG
  - RC_4: number of values between AVG and $AVG + STDEV$
  - RC_5: number of values between $AVG + STDEV$ and MAX
  - RC_6: number of values above MAX

  With these counters we can detect, eg, the insertion of an unusually large number of small values. Again, RC_i raises an alarm if the absolute or relative value changes too much.
- $CATC_i$: Category Counters.
  Defined analogously to RC_i, the category counters divide the extension of ASCII string-type and date/time-type attributes into several partitions and monitor the population in each partition. We use the following categories:
  - For date/time-type attributes: Month, day, weekday, hour, minute, second and millisecond
  - For ASCII string-type attributes: the fraction of letters, digits and other characters

  With these counters we can detect, eg, the insertion of an unusually large number of date data with the month January. An abnormal absolute or relative change results in an alarm.
- ZLC: Zero-Length String Count.
  It holds the number of non-NULL ASCII-strings with a length of zero. This is important because these instances have to be excluded from the computation of letter, digit and other character fractions. Again, an abnormal absolute or relative change results in an alarm.
- PBC: Positive Bit Count.
  It holds the number of non-NULL values in a bit-type attribute's extension with positive bit value. It reports an anomaly if the positive bit count increases or decreases absolutely or relatively too much.
- For each of the values RC_i, $CATC_i$, ZLC and PBC there is an additional value that describes the ratio of this value to NNC.
  - $RR_i = RC_i/NNC, i = 1, \ldots, 6$: Range Ratios
  - $CATR_i = CATC_i/NNC$: Category Ratios

- $ZLR = ZLC/NNC$: Zero-Length String Ratio
- $PBR = PBC/NNC$: Positive Bit Ratio

An abnormal absolute or relative change results in an alarm.

There are a few subtleties in the computation of the deviations. The rule for NNC is simple:

$$\Delta NNC = NNC_2 - NNC_1$$

and

$$r\Delta NNC = \frac{\Delta NNC}{NNC_1}$$

But an analogous computation of the relative change of the MIN value does not yield the expected result. Consider an attribute that stores the year of birth of students. Suppose that the minimum value is 1972 and maximum is 1982. If a senior student born in 1932 joins the group, then $(1932 - 1972)/1972 = -0,02$, which is negligible. More important and really anomalous is the fact that the new minimum extended the range of this attribute by 400%. Thus, we here use the formula $(1932 - 1972)/(1982 - 1972) = -4$, ie:

$$-r\Delta MIN = -\frac{MIN_2 - MIN_1}{MAX_1 - MIN_1}$$

Finally, let us take a look at the average. A comparison of the old and new average values does not reveal an important anomaly. Suppose that a small number of anomalous tuples is inserted. Then $AVG_1 - AVG_2$ is likely to remain inconspicuous. To detect this anomaly we must compare AVG_1 with the average of these new tuples, ie with

$$\frac{1}{\Delta NNC} \sum_{i=NNC_1+1}^{NNC_2} x_i$$

These observations apply also to the standard deviation (but yield a much more complex formula).

The subsequent table summarises the use of the various parameters for the data types.

	OC	NNC	NNR	MIN,MAX	RANGE	AVG,STDEV	RC_i	RR_i
Numeric		X	X	X	X	X	X	X
ASCII string		X	X	X	X	X	X	X
Binary		X	X	X	X	X	X	X
Date and Time		X	X	X	X	X	X	X
Bit		X	X					
Unique ID		X	X					
Relation	X							

This approach yields the best results with growing-only relations. If updates and deletions of a large number of tuples are permitted, anomalous operations may remain undetected. To give an example, let us monitor only the number

	CATC	CATR	PBC	PBR	ZLC	ZLR
Numeric						
ASCII string	X	X			X	X
Binary					X	X
Date and Time	X	X				
Bit			X	X		
Unique ID						
Relation						

of tuples in a relation, and suppose that the normal behaviour of this relation is a growth of ten tuples between two runs of the ADS. Here we cannot detect an insertion of a million of tuples that is followed by a deletion of a million of tuples.

3.2 Anomaly Detection Based on Δ-Relations

The concept of Δ-relations is an extension to our approach that can detect various anomalies on relations regardless of the number of inserted, updated or deleted tuples. The Δ-relations record all changes to a relation, including old values of updated tuples and deleted tuples. Δ-relations can require a lot of storage, but they provide a precise and comfortable means to discover several anomalies. For example, an unusually high number of inserted tuples followed by a similar number of deleted tuples can now be detected. Moreover, we can identify the tuples that exceed a threshold.

Anomaly detection with the support of Δ-relations works as follows. For every relation that should be monitored four Δ-relations are created:

- INS: stores inserted tuples
- DEL: stores deleted tuples
- UPB: stores updated tuples before the update
- UPA: stores updated tuples after the update

There are two ways of using the data in the Δ-relations.

The first way computes a fictitious 'After'-state. Here we take all tuples from one Δ-relation, eg INS, and assume that only changes to these tuples have been made to the relation in its state before the current run of the ADS. This gives us a fictitious relation state after the run, which then can be compared to the before-state with the methods described in the previous section.

Δ-relations require some modifications to the way of detecting anomalies. On the one hand, it is now very simple to deal with values that record numbers of tuples, since we can directly compare the numbers in the relations. For example, we can count the number of deleted tuples by counting the number of tuples in the DEL relation (and omit a comparison of old and new states). The computation of MIN, MAX and $RANGE$ remains unaltered, and that of AVG and

$STDEV$ is straight forward. On the other hand, the computation of ratios is more complex. For example, we now have

$$|\Delta O| = |NNC_2/OC_2 - NNC_1/OC_1|$$

All the reference values can now be computed also for the Δ-relations and used in the anomaly detection.

The second way of using the data in the Δ-relations is direct comparison. Here the reference values for a Δ-relation are directly compared to the reference values of the associated relation in the state before the changes. Unfortunately, this approach is much too prone for erroneous alarms, in particular, if the Δ-relations are small. To give an example, consider an attribute of type bit. Suppose that 60% of values in its extension are 1 and that only a single element with the value 1 is inserted. This results in value of $PBR = 100\%$ for Δ-relation INS, which deviates by 40% absolutely from the value of PBR for the associated relation in its state before the change and by 66.7% relatively. Both deviations are like to raise a false alarm.

3.3 The Determination of Threshold Values

We suggest three ways for the determination of suitable thresholds.

We can set all thresholds manually to values that are dictated by our experience with the database. While it may be easy to decide on the relative thresholds, eg 5% or 10%, a careful determination of absolute thresholds can be very labour intensive – our system may require far more than 100 such values.

Secondly, we can assume that the initial database state, viz the state that is given to our anomaly detector for the first time, is normal. Then compute all reference values for the first time and derive the thresholds from them.

And, lastly, we can conduct a training phase with presumably regular activities and compute all reference values several times. Then consider the development of the reference values and determine the thresholds on these grounds. Still, we recommend a manual graphical analysis of this development to verify the plausibility of the thresholds.

4 Anomaly Detection in the User Interaction

Anomaly detection in the user interaction can make use of the same methods as anomaly detection in the database extension for the following reasons.

During the analysis cycle the ADS collects user interaction data with the auditing tools of the DBMS. This audit data consists of several sets of values that can be stored as tuples in a database table; we call this table the TraceTable. A single tuple represents a single user operation. Since the TraceTable is an ordinary relation with several attributes, the extension of which grows between two runs of the ADS, we can apply the analysis methods for the database extension to it. Note that we do not need to use Δ-relations.

In the auditing phase the following elements of a user operation are audited and stored in TraceTable:

- the user name
- the SQL command string
- the command's start time
- the command's duration
- the CPU time used by the command
- the number of affected tuples
- SQLCommandClass, a special attribute that classifies the SQL command with one of the follwing values:ADMIN, DDL, PRIVILEGE, DML, READ, EXEC and NULL.

During each run of the ADS all reference values are computed for this relation and compared in the familiar fashion with the corresponding old values.

The SQLCommandClass and the start time represent the part of the user interaction that is controlled by the user. Thus, these attributes reveal an anomaly in the user input.

SQLCommandClass is of type ASCII string and all applicable reference values are computed for it. However, this attribute can take only one of seven values and, therefore, $CATC_i$ and $CATR_i$ provide the most valuable hints to an anomaly. The attribute start time is of type datetime and is treated accordingly. An example of easily detectable anomalies are: a user submitted today too many DDL-commands or at an unusual time of day.

The system behaviour is the DBMS reaction to the user input. To check it for anomalies, we must analyse the duration, CPU time and number of affected tuples. All of them are of a numerical data type; thus, the reference values for anomaly detection in numerical attributes are applied to them.

5 Misuse Detection in the Database Scheme

To develop a comprehensive ADS for relational databases, we must also consider anomalies in the database scheme. Our approach relies on a library of attack signatures and is in fact a misuse detection system.

We first examine all database objects (MS SQL Server has 12: database, default, function, index, privilege, procedure, rule, schema, statistics, table, trigger and view) and all operations on them. Then we classify them with respect to the threat that they pose to a database and store a list of dangerous commands. The commands issued by users are compared to this list.

Let us take a look at a few dangerous commands. The database object is clearly critical. The drop database command is dangerous, for it does not only deletes the database from the system catalog but also the database files stored on disk. We do not consider the alter database command dangerous, because it can delete only empty database files. An index for a table is critical. A single drop index command can severely degrade availability, but also a large number of create index commands can affect performance.

6 A Brief Example of Anomaly Detection in a Database

We now present some screen shots of our ADS. It is applied to an example database, which is populated with data of about 300 CDs with their 4000 songs.

Figure 1 shows data that relate to the length of the songs' titles. The diagram visualises the distribution of the lengths. The Overall Watch section shows reference data for the old extension. The Delta Watch section shows the changes that occurred to the database in the meantime. We see that the standard deviation has increased by nearly 5%, which is due to an increase of the population in the R5-area by nearly 27%. In Figure 2, the Overall Watch section shows the data of the current extension. Note the dotted line in the R5-area, it depicts the value of the old extension. Figure 3 compares in the diagram the old values, represented by the dotted lines, to the new values, shown in solid colours. Here we can easily spot changes to the MIN, MAX, AVG and $STDEV$ values, as well as changes in the numbers of tuples in the respective areas. This provides a visual indication of anomalies.

Figure 4 analyses the fraction of letters in the songs' titles. We see that the numbers remained fairly the same and give no reason for a concern.

Figure 5 shows an analysis of the distribution of months in a date type attribute.

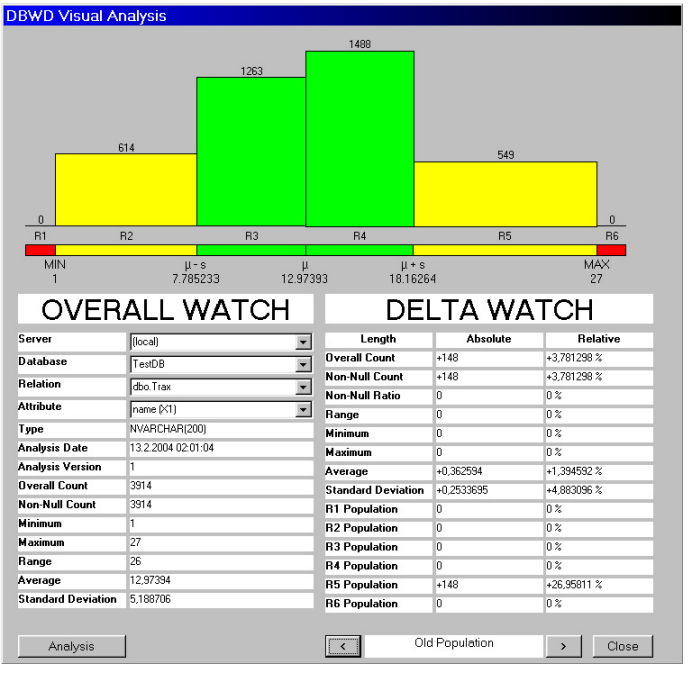

Fig. 1.

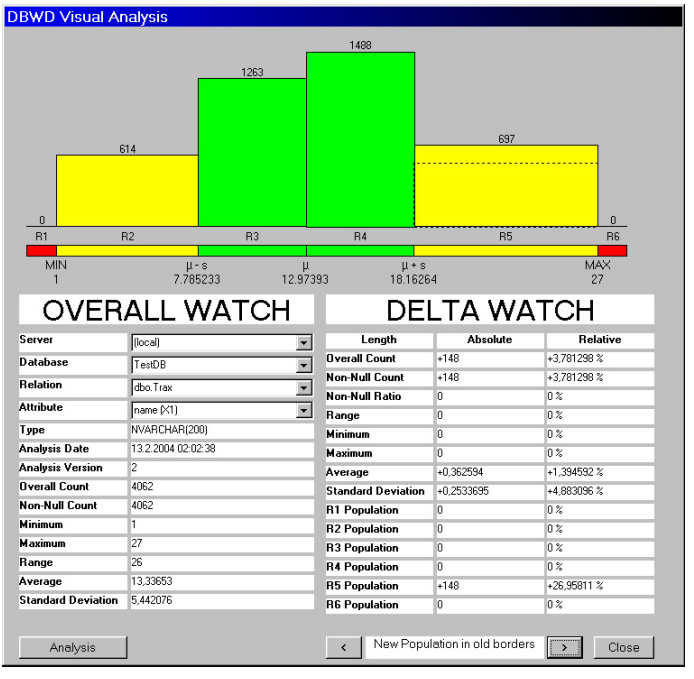
DBWD Visual Analysis
1488
1263
614
697
0
0
R1
R2
R3
R4
R5
R6
MIN
1
μ - s
7.785233
μ
12.97393
μ + s
18.16264
MAX
27
OVERALL WATCH
DELTA WATCH
Server (local)
Database TestDB
Relation dbo.Trax
Attribute name (X1)
Type NVARCHAR(200)
Analysis Date 13.2.2004 02:02:38
Analysis Version 2
Overall Count 4062
Non-Null Count 4062
Minimum 1
Maximum 27
Range 26
Average 13,33653
Standard Deviation 5,442076
Length Absolute Relative
Overall Count +148 +3,781298 %
Non-Null Count +148 +3,781298 %
Non-Null Ratio 0 0 %
Range 0 0 %
Minimum 0 0 %
Maximum 0 0 %
Average +0,362594 +1,394592 %
Standard Deviation +0,2533695 +4,883096 %
R1 Population 0 0 %
R2 Population 0 0 %
R3 Population 0 0 %
R4 Population 0 0 %
R5 Population +148 +26,95811 %
R6 Population 0 0 %
Analysis
New Population in old borders
Close

Fig. 2.

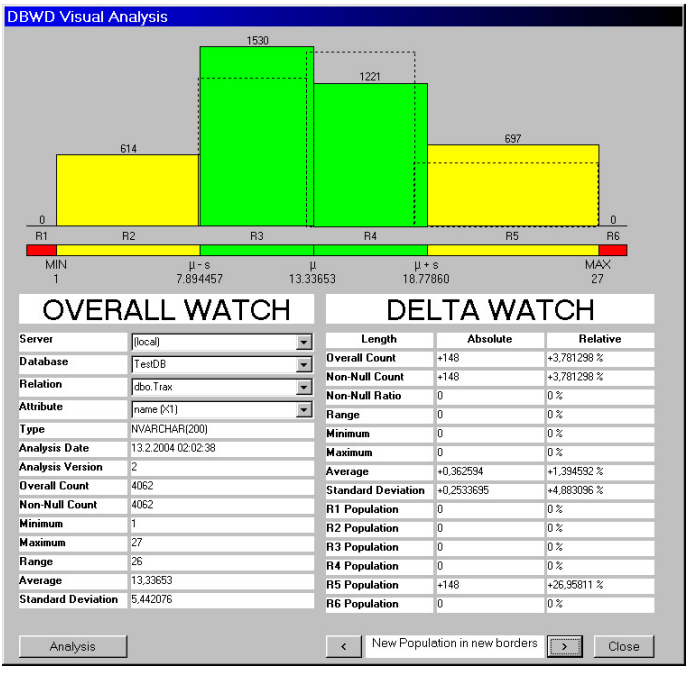
DBWD Visual Analysis
1530
1221
614
697
0
0
R1
R2
R3
R4
R5
R6
MIN
1
μ - s
7.894457
μ
13.33653
μ + s
18.77860
MAX
27
OVERALL WATCH
DELTA WATCH
Server (local)
Database TestDB
Relation dbo.Trax
Attribute name (X1)
Type NVARCHAR(200)
Analysis Date 13.2.2004 02:02:38
Analysis Version 2
Overall Count 4062
Non-Null Count 4062
Minimum 1
Maximum 27
Range 26
Average 13,33653
Standard Deviation 5,442076
Length Absolute Relative
Overall Count +148 +3,781298 %
Non-Null Count +148 +3,781298 %
Non-Null Ratio 0 0 %
Range 0 0 %
Minimum 0 0 %
Maximum 0 0 %
Average +0,362594 +1,394592 %
Standard Deviation +0,2533695 +4,883096 %
R1 Population 0 0 %
R2 Population 0 0 %
R3 Population 0 0 %
R4 Population 0 0 %
R5 Population +148 +26,95811 %
R6 Population 0 0 %
Analysis
New Population in new borders
Close

Fig. 3.


DBWD Character Analysis
Old Distribution
New Distribution
Absolute
Ratio
57
0
Category
0..10 %
10..20 %
20..30 %
30..40 %
40..50 %
50..60 %
60..70 %
70..80 %
80..90 %
90..100 %
Non-Null Count Old
70
Non-Null Count New
71
Abs. Old
Abs. New
Diff
Diff (%)
Ratio Old
Ratio New
Diff
Diff (%)
Letter
Close

Fig. 4.

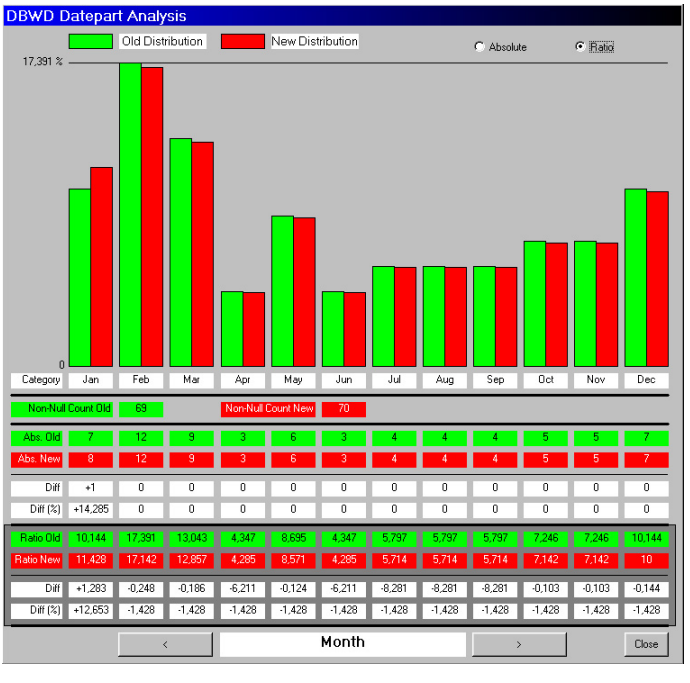
DBWD Datepart Analysis
Old Distribution
New Distribution
Absolute
Ratio
17,391 %
0
Category
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Non-Null Count Old
69
Non-Null Count New
70
Abs. Old
Abs. New
Diff
Diff (%)
Ratio Old
Ratio New
Diff
Diff (%)
Month
Close

Fig. 5.

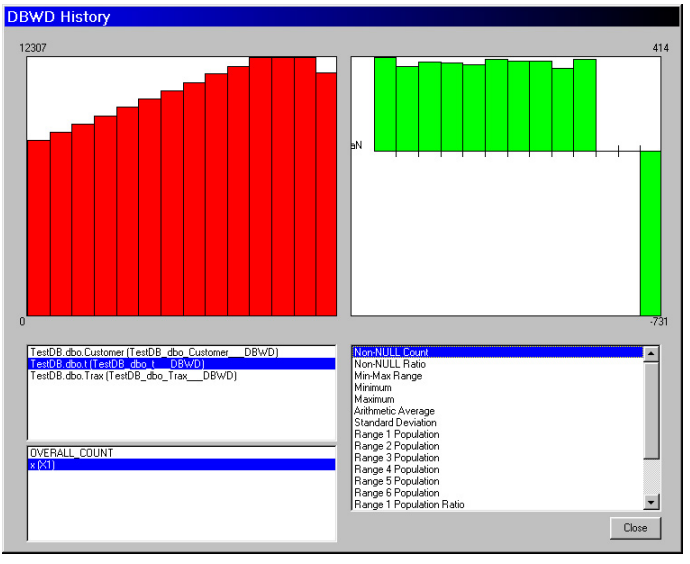

Fig. 6.

And, lastly, Figure 6 shows a longer history of a reference value, the non-Null count. We see in the left panel the absolute numbers and in the right panel the corresponding differences. We note that a continuous increase is followed by an abrupt decrease. The lower right panel allows the selection of numerous parameters.

7 Conclusion

In this work we have presented an anomaly/misuse detection system for relational databases. It can monitor the extension, the user interaction and the scheme. For each data type of the attributes there are numerous values which capture characteristic properties of the attribute's extension. For predominantly growing relations these reference values can help to detect a variety of anomalies. This technique can be applied to Δ-relations to detect even more anomalies in arbitrary relations, however, their space requirements must be carefully calculated. Reference values are also the key to anomaly detection in the user interaction. The ADS can detect many anomalies in the user input and in the reaction of the DBMS. Lastly, we sketched a misuse detector for the database scheme.

The ADS is implemented for the MS SQL Server 2000 and can be applied to any existing database.

On the conceptual side, we would like to examine in future the analysis of groups of attributes with data mining techniques and the suitability of our ADS for real-time detection.

References

1. Axelsson, Stefan. 'Intrusion Detection Systems: A Survey and Taxonomy'. *Technical Report 99.15* Dept. of Computer Engineering, Chalmers University of Technology, Sweden, 2000.
2. Burgess, Mark, Hårek Haugerud, Sigmund Straumsnes and Trond Reitan. 'Measuring system normality'. *ACM Transactions on Computer Systems* 20.2(2002):125-160.
3. Chung, Christina Yip, Michael Gertz, and Karl Levitt. 'DEMIDS: A misuse detection system for database systems'. *IFIP WG11.5 3rd Working Conference on Integrity and Internal Control in Information Systems*, pp 159-178. Kluwer Academic Publishers, 1999.
4. Gao, Debin, Michael K. Reiter and Dawn Song. 'Gray-box extraction of execution graphs for anomaly detection'. *11th ACM Conference on Computer and Communications Security*, pp 318-329. ACM Computer Press, 2004.
5. Gertz, Michael. 'Data Content Monitoring for Security, Integrity and Availability: A Mission-Critical Line of Defense'. *IICIS 2002: IFIP WG11.5 5th Working Conference on Integrity and Internal Control in Information Systems*, pp 189-201. Kluwer Academic Publishers, 2003.
6. Lee, Sin Yeung, Wai Lup Low, Pei Yuen Wong. 'Learning Fingerprints for a Database Intrusion Detection System'. *ESORICS 2002: 7th European Symposium on Research in Computer Security.* LNCS vol 2502, pp 264 - 279. Springer-Verlag, 2002.
7. Lee, Victor C.S., John A. Stankovic, Sang H. Son. 'Intrusion Detection in Real-time Database Systems Via Time Signatures'. *RTAS 2000: 6th IEEE Real Time Technology and Applications Symposium*, pp 124-133. IEEE Computer Society Press, 2000.
8. Michael, C.C., and Anup Ghosh. 'Simple, state-based approaches to program-based anomaly detection'. *ACM Transactions on Information and System Security* 5.3(2002):203-237.