

Stay Alert to your Data Integrity with Exception Reports

Peter Robson

Oracle Database Specialist
e-mail: Peter.Robson@JustSQL.com

“TO MANAGE YOU MUST FIRST BE ABLE TO MEASURE”

Abstract

Exception reports are a familiar technique to the Oracle dba (checking runtime parameters etc), but they are particularly important when used to test for any conditions that compromise the integrity and consistency of the data in the database. No matter how fast and efficiently Oracle actually runs, if it is holding corrupt data – the plot has been lost! Therefore exception reports are a major component in the maintenance of data quality.

The condition of the data in the dbms (as distinct from the dbms function itself) can be monitored by: building a comprehensive metadata system around the database, running exception reports on the integrity of this metadata, and testing the veracity of the metadata references to the actual underlying data.

Alert systems have been built that run over 150 tests every night, with the results e-mailed to key personnel, and logged into persistent store in the database itself. These alert reports are distributed by e-mail and intranet page publishing, using automated techniques based on two key underlying dictionary tables.

The presentation will describe the various types of alerts being executed, and demonstrate the scripts used to run, process and distribute them.

1. Background

One of the major characteristics of a relational database management system is the crucial interdependence that exists between the various objects within a schema. Oracle is no exception in this regard. Even in a schema with only one table, there will be associated objects with that table, such as a primary key constraint (not always present, but really obligatory in a relational model), as well as various privilege settings.

As the overall model starts to grow in size, additional tables will rapidly appear, each with their own associated objects, together with foreign key constraints enforcing the inter-dependence between these various tables. The full range of objects involved in a schema (tables, constraints, indexes, triggers, etc) all play a part in defining the overall model. It is the strictures of the relational modeling methodology that ensures that all these parts co-exist together, and contribute to the overall coherence and integrity of the model.

The subject of this paper is concerned with the systems which can be built to ensure that the integrity of the data model is maintained. With a very complex schema (here the author refers to the model in which these ideas have been developed, where something like 7500 objects exist, of which 2500 are tables), it becomes essential to adopt metrics of some sort to provide the assurance that the integrity of the model is being maintained. This activity is based on a working Oracle database system within a heterogeneous scientific environment, where the data diversity is considerable. Efforts to take a corporate view across such a wide diversity of data absolutely requires that metrics be applied across the metadata.

In addition to the basic Oracle provision for building a relational model, the author has built an extensive metadata layer around this already complex model. This metadata defines and documents such things as the personnel responsible for the various objects in the schema, descriptions of all the database objects, as well as defining the naming conventions for all objects in the schema. Because this metadata layer conforms to standard relational modeling practice (e.g. metadata components are regarded as attributes of the core data components, and thus modeled accordingly), one can take advantage of the fundamental orthogonality of the relational model to validate the presence and function of the various components of the metadata layer.

The integrity of the system has already been identified as an important reason for running alert metrics. This raises the whole matter of quality assurance. Ultimately, the user (or customer, they amount to the same thing) must be assured of the quality of the data. This can only be done if an adequate metadata layer is integrated into the database, and done so in a strictly formal manner, and further, that it is continually checked to maintain assurance that the system conforms to design.

This paper will describe the major aspects of the metadata system, and how the database is monitored on a daily basis.

2. Database Metadata

The author has already presented to the PLOUG conference on this subject ([Rob00]), but a brief summary of the major points will be provided here.

- Business rules define the reason for the existence of an object in the database. They must be documented.
- All database objects must be documented, and 'responsible persons' assigned to the management of each object.
- All database objects must be named according to corporate standards, these standards are defined for the various categories of objects (e.g. tables, indexes, constraints, views, triggers, etc).

- All tables must be defined by a primary key, be documented, and assigned a manager.
- Important corporate attributes must be documented and defined.
- All table access privileges must conform to corporate standards
- All tables (and related database objects) must be assigned to one or more ‘Databanks’. (Databanks are defined as subject groupings of database tables and associated objects).
- All databanks must be assigned to a corporate project activity.

Against this variety of metadata, over 150 individual alerts have been constructed to run against the database every night. These perform two functions – firstly, where an exception condition is detected, the data reflecting that exception is extracted into a report file, and e-mailed to the dba and relevant data managers. Secondly, a numerical score of that exception count is logged to a database table, and held in permanent store. This record then allows a historical review of exception conditions, how long they remained extant, and how the frequency of the exception varied over time.

3. Categories of Exception Reports

The exceptions themselves have been classified into some half dozen different categories:

Metadata alerts test all database objects for conformance to corporate standards. This includes many of the issues already listed above.

Permission alerts notify where tables have not been assigned their correct privileges. Typically, this would identify tables that did not have the default ‘public’ select, where ‘public’ is defined as the total scientific staff (all scientific data in the organisation is regarded as shareable, with rare confidential exceptions). Tests are also run to ensure that each table manager has the appropriate level of access to enable them to manage ‘their’ tables commensurate with their responsibilities.

Replication alerts monitor the processes that are required to enable the overnight replication procedures to run between the various geographically dispersed Oracle instances.

Structural alerts are defined as those that seek out invalid or disabled objects, most typically views, constraints and triggers. Also included in this category are tests to ensure that objects have been placed into their correct tablespaces.

Data Error alerts are used to identify actual bad data, most particularly the inclusion of non-printable characters in text or description fields. A list of table columns are identified that are susceptible to this type of corruption, and these are scanned every night for instances of data error. These corruptions are most typically caused by staff entering data using the ‘cut and paste’ method from Microsoft Word documents (and similar) via Access front-end applications.

Process alerts are high priority instances of actual data errors that would impact on subsequent data entry. These are flagged up for immediate attention.

Finally, there are a suite of *Summary* alerts that are e-mailed to the various responsible persons, in which the total alerts for that night are summed, compared to the runs from the previous night, and subdivided according to the person most able to effect the corrections.

4. Alert Construct in Detail

All of these alerts are driven by a cron job in the early hours of each morning. This job kicks off a SQL*Plus session against Oracle, and runs a suite of individual scripts within one large script file. Each script tests the database for conformance to the relevant alert condition.

Each alert script is defined with an alert number, which accompanies the reporting of the alert right through into persistent storage, and on to the e-mail alerts. When a new alert is required, two scripts are written into one single text file. These scripts perform the two functions outlined above, firstly to extract into an alert text file those database rows which show evidence of an exception condition, and secondly, to place into persistent store within the database the numerical count of that alert condition.

Once this composite script is written, either by an analyst, or the dba, it is placed into a designated public subdirectory on the central (Windows) file system. During the overnight batch run, this script is automatically moved, using FTP, to a storage area on the Unix environment from which the alert tests are run. Here every individual alert is concatenated into one single sql script file, which then becomes the script file run by the cron job. This concatenation takes place every night, to ensure that the composite script file always includes the very latest additions to the alert script inventory.

The alert script uses several Oracle tables. Each alert is documented in the table 'Dic_Alerts'. Each alert is, as described above, classified into one of six categories – defined in table 'Dic_Alert_Category'. Each alert is assigned to an individual whose task it is to correct any incidence of an alert condition. This dependency is reflected in the table 'Alert_Managers'. See Figure 1 which illustrates the entity-relationship between these tables.

ER Diagram for Excecion Alert tables (MHA)

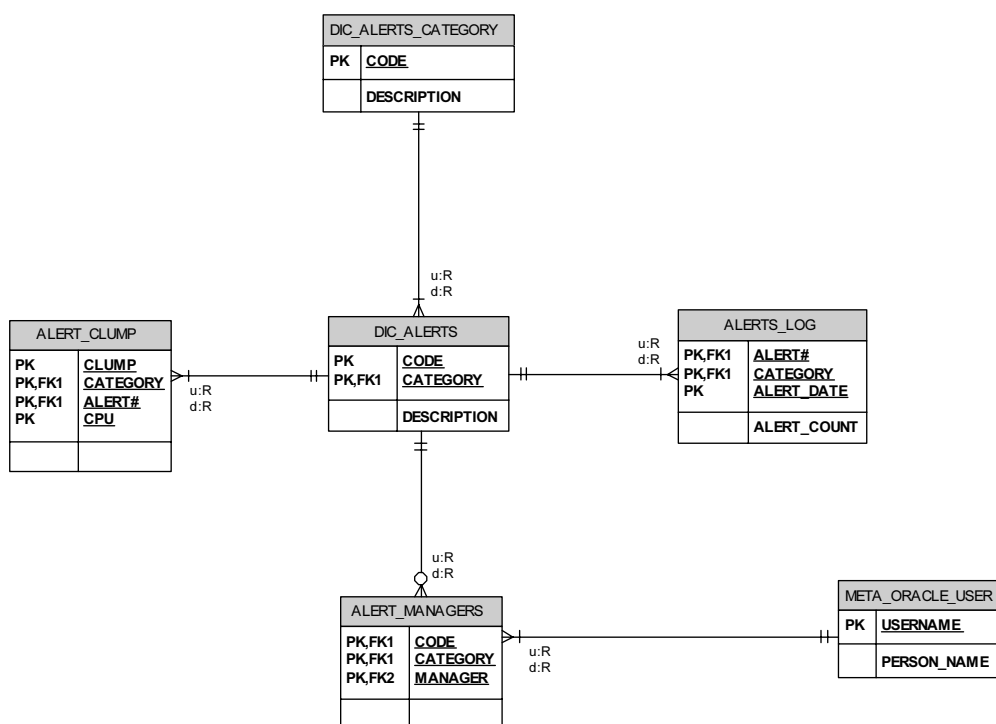


Figure 1. Entity-Relationship diagram for the Alert support tables

As well as running the alert tests themselves, the job uses SQL to construct the unix shell script which a) tests for the presence of data in an alert spool report, and b) if data is present (eg alert data was found) the unix mail commands are created. These mail shots are despatched to the persons whose names appear in the table 'Alert_Managers'.

The construct of the whole system requires the minimal input from the developer when introducing a new alert. The following are required:

1. Build the two scripts to check for alert conditions
2. Place that single script file into a designated public area
3. Document the alert in the table Dic_Alerts
4. Assign the alert to one Category in table Dic_Alerts_Category
5. Assign one or more managers to monitor the alert in table Alert_Managers
6. Optionally, 'clump' the new alert in table Alert_Clump (see below)

From this point on, the process is fully automated.

The only table not discussed is Alert_Clump. This provides a simple way of grouping, or 'clumping' alerts into meaningful collections, outwith the strict definition of a category. Thus clumps of alerts may relate to 'tables', 'privileges', 'trigger', etc, simply as a means of allowing anyone to check all alerts which refer to, for example, all triggers on the system, irrespective of their category.

The alert dictionary table (Dic_Alerts) has the following structure:

```
SQL> desc dic_alerts
Name                               Null?    Type
-----
CODE                                NOT NULL NUMBER
DESCRIPTION                          VARCHAR2(200)
CATEGORY                             VARCHAR2(20)
```

The *code* attribute is the alert number, and '*category*' is as defined and described above. The alert number appears in the field Code in this table, but as '*Alert#*' in the table 'Alerts_Log', which is where the alert totals are accumulated day by day. 'Code' is the standard attribute name for a dictionary primary key field.

This construction and inter-dependency of the various components of the alert system are best illustrated in diagrammatic form. The next section will describe in detail exactly how the system works.

5. The Structure of the Alert System

Several inter-connected activities are bound up in the overall alert system. Figure 1 presents an overview of the system.

1. SQL scripts generate listings of all offending items in the database where an alert condition is violated. The output of each alert test is a single alert text file (eg 'alert_3.txt', 'alert_12.txt', alert_132.txt', etc.).
2. Further suites of scripts generate a count for each alert condition, and then store that count result back into the database. The alert text files are copied to the web server.
3. All exception condition files that have been generated by the alert tests are e-mailed out to the appropriate recipients. This activity cannot take place until actions 1 and 2 have completed.
4. A report generator queries the database for a list of all categories of alert conditions, then a list of all alerts by category. Because these items are defined in the database, whenever a new category or alert is added (or indeed removed), that change will be automatically reflected in the composition of the report pages. These pages are then copied to the web server.

5. When a user queries the alert report pages, any alert text files that are relevant will be available for viewing within the appropriate report page because they have already been copied to the web server (see #2 above).

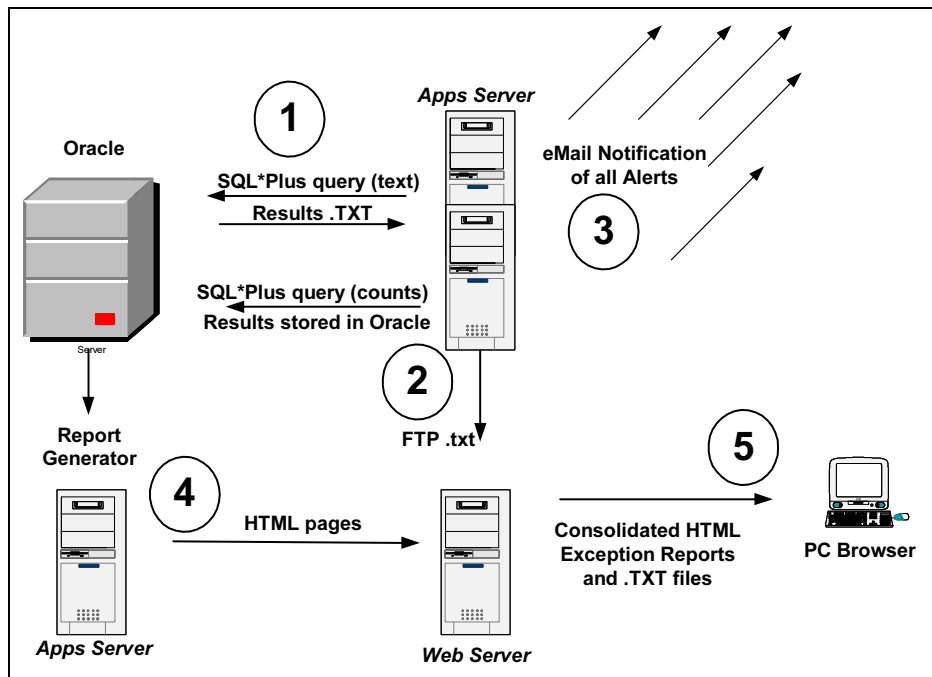


Figure 2 – schematic of the Exception Reporting process.

The schematic Figure 2 illustrates the process dependencies involved in generating the alert reports. The bold numbers in circles indicate the order in which actions take place and refer directly to the numbered bullet points above. Actions 1, 2 and 3 are all under the control of a cron job. Action 4 is a scheduled activity on an NT workstation (described as an Apps Server for the purposes of this illustration).

6. Example Alert Scripts

The construction of this alert system will be demonstrated by following example scripts through the alert process. There is considerable variation in complexity, dependent on the nature of each individual alert, but the basic construct is similar throughout.

Each alert establishes two populations – first, the total population for the object being tested, and then second, the population of those objects that are known to conform to the defined corporate standard. The result presented in the alert report is the subtraction of the second population from the first – thus providing a list of all those objects that do not conform to standard. The operation is achieved through repeated use of the SQL set-based function ‘minus’. In the ideal situation, both populations would be identical, thus giving a zero result to the subtraction.

An example of a simple alert would be the test that checks whether or not a table has been documented. All database objects are listed in metadata tables, the key table for this function being ‘metadata_objects’. This holds the name of the object, a description attribute, and the name of the relevant data manager. Actually, each object has two managers – one who is cognizant with the data, and the second who is responsible for the design of the table and implementation of any applications. These roles may be regarded as the ‘logical’ and ‘physical’.

This alert has the description ‘Tables in the corporate schema, but not documented in the metadata’. The important metadata table to note is defined as follows (the primary key fields being the first four):

```
SQL> desc meta_objects
Name                                Null?    Type
-----
OWNER                                NOT NULL VARCHAR2 (30)
OBJECT_NAME                          NOT NULL VARCHAR2 (30)
OBJECT_TYPE                          NOT NULL VARCHAR2 (13)
CPU                                   NOT NULL VARCHAR2 (1)
DESCRIPTION                          VARCHAR2 (2000)
DB_ADMIN                             VARCHAR2 (30)
AP_MAN                              VARCHAR2 (30)
```

Note that the attribute ‘cpu’ is used here, because this metadata system applies to several instances of Oracle. Purely for reasons of clarity, this constraint will not be used in these example scripts. This alert will check that all objects in the corporate schema that are tables, have a value in the description field of the metadata table. Whether that description is correct or not is an entirely different matter, of course. The script is as simple as this:

```
select object_name from user_objects
where object_type = 'TABLE'
minus
select object_name from meta_objects
where object_type = 'TABLE'
and description is not null ;
```

... and will provide a list of any offending tables. (The tests are run against ‘user_objects’ rather than all_objects or dba_objects because they are executed within the corporate schema.)

A count of this result is also required. This is achieved by populating an ‘alert count’ table, that has the following definition:

```
SQL> desc alerts_log
Name                                Null?    Type
-----
ALERT#                              NOT NULL NUMBER
ALERT_DATE                          NOT NULL DATE
ALERT_COUNT                          NOT NULL NUMBER
ALERT_REF                            NOT NULL NUMBER
```

The attributes are as follows:

- Alert# is the alert number (exactly the same as ‘code’ in table Dic_Alerts);
- Alert_Date is the date that the alert count is made.
- Alert_Count holds the count value itself.
- Alert_Ref is a count of the total population that is being examined. This enables a percentage figure to be calculated, to indicate what proportion of the total objects being examined are failing the exception test. This point will be enlarged on subsequently.

The script to populate the count table for the current example has this general form:

```
insert into alerts_log
(alert#,alert_date,alert_count,alert_ref) values
(34, sysdate,
(
select count(*) from user_objects
where object_type = 'TABLE'
and object_name in
(
select object_name from user_objects
where object_type = 'TABLE'
minus
select object_name from meta_objects
where object_type = 'TABLE'
and description is not null
)
),
(select count(*) from user_objects where object_type = 'TABLE')
);
```

Note how the four values required for the count are assembled by using two distinct scripts embedded within the insert component. The last sub-script generates a count result for the total population being sampled, e.g. user_objects of type 'TABLE'. The inserted row, when retrieved from the table 'alerts_log', is as follows:

ALERT#	ALERT_DATE	ALERT_COUNT	ALERT_REF
34	Feb 17, 2004 14:45 49	20	652

This result indicates that there were 20 tables out of 652 already lodged in the metadata table that do not have a description. These 20 results are listed to a spooled text file by the first alert script discussed, and then e-mailed out to the relevant persons. The e-mail command lines are generated on the Unix platform, by an Oracle script, that obtains all the information required from the alert dictionary and alert result tables. An alert file is generated for every single alert text, but only those with a result are mailed out. A 'grep' command (on Sun Solaris) used with a conditional test, identifies only those alert result files that have the typical Oracle heading output underline characters ('----') present – if a result heading is not present, these characters will NOT be present in the alert file, no exception data will have been retrieved, and therefore a mail shot will not take place. For reasons of space, this line will be listed over three lines, but it is in fact one single line without any <cr><lf> controls:

```
grep ^---- $ALERT_SUBDIR>alert_34.txt' && /usr/ucb/mail -s
"ALERT 34 - Tables in the corporate schema, but not documented in the metadata
" pgro < ALERT_SUBDIR>alert_34.txt'
```

The alert result text file is 'alert_34.txt', and the e-mail recipient is 'pgro'. The environment variable '\$ALERT_SUBDIR' refers to the location of the various alert text files.

There is a final stage to this alert reporting process, as the alert text files are automatically picked up by an FTP job that checks every hour for additions to the sub-directory holding these results. When new report files are detected, they are shipped across to a subdirectory on the local intranet

web system. In their turn they are picked up by a report writer that produces alert reports for the intranet every night. This is the last job to run, and kicks off at 6.0am each morning.

The end result of these activities is that alert exception conditions are mailed out to key personnel every morning, and a total summary of all exception conditions is posted to the intranet site just before the start of the working day. These results are grouped by the half dozen different categories already described above. Such is the construction of these report files, that readers can drill down to the individual report from the highest level of the intranet technical metadata pages.

7. The Current Alerts

The previous section described in detail how one particular alert would function. The following is an inventory of the various types of alerts, grouped according to their category. This is not an exhaustive list, as there are some alerts that are highly specific to the site on which they were developed, but the range and scope of the alerts used will be clear.

Each alert is described in terms of a non-valid or non-legal condition.

Table 1. Alert Categories and their Individual Alerts

Alert Category	Alert Description
<ul style="list-style-type: none"> • Metadata 	Business Rules incomplete Databanks without a corporate project code Databanks lacking a description Databanks lacking a Manager Tables / Views / Triggers / Indexes / Constraints incorrectly named Tables / Views / Triggers / Indexes / Constraints not present in Metadata Tables / Views / Triggers / Indexes / Constraints not documented Tables / Views / Triggers / Indexes / Constraints not assigned a Databank Tables / Views / Triggers / Indexes / Constraints lacking Managers Moribund tables still extant Orphaned audit tables (master missing) Retired staff still with database privileges
<ul style="list-style-type: none"> • Data Errors 	Text fields with multiple adjacent single quotes Text fields containing non-print characters (ascii 10 and 13) Column mismatch between master and audit tables High incidence of null values in attributes Incorrect data value in specified fields
<ul style="list-style-type: none"> • Permissions 	Missing permissions for table managers Public select missing for shareable tables Incorrect permissions on audit tables Unauthorised Alter and Index permissions Incorrect Reference permissions

Alert Category	Alert Description
<ul style="list-style-type: none"> Replication 	Incorrect tables flagged for replication Tables flagged for replication without a complimentary audit table Tables flagged for replication missing correct triggers Failure of post-replication comparison tests
<ul style="list-style-type: none"> Structural 	Tables lacking a Primary Key PK update prevention triggers missing Triggers with status Disabled Constraints with status Disabled Primary Key attributes with multiple definitions across tables Incorrect tablespace allocation for tables / indexes Table names comprising lower case alpha characters Invalid triggers

Several individual alerts are indicated by only one entry above. For example, tablespace allocation is checked for all tables and then for all indexes, each as a distinct activity. Further, because these checks are being applied to several geographically remote instances of Oracle, these same tests are run separately against each instance.

8. The Alert Counts

As well as the alert violations for each day being detected and distributed through e-mail, the count of these alerts is maintained in the table 'alerts_log'. This enables long-term trends in alert violations to be detected, as well as providing the ability to monitor progress in correcting outstanding exception conditions.

Because the alert counts are held with the total count for the population being examined for alerts, it is an easy matter to establish a query in which the percentage violation per population is presented. For the purposes of example, a small sub-set of alert counts will be illustrated.

The following script will present the exception count for each alert, the total population examined, and the percentage errors:

```

select alert#,
       substr((
         round(100 - (((alert_count)/(alert_ref))*100),2)||'%' ,1,12) " Alert Per-
cent",
       alert_count,alert_ref,
       description
from alerts_log al, dic_alerts d
where alert# = code
and alert_date > sysdate - 1
and alert_count <> 0
order by alert# ;

```

Note that it assumes a query ran 'yesterday' (e.g. 'sysdate - 1') to ensure that the computation only applies to data accumulated during the current day. A typical result for a sample of rows would appear as follows:

Alert#	Percent Correct	Error_ Count	Alert_ Ref	DESCRIPTION
4	86.11%	680	4897	Tables lacking a description
7	86.54%	659	4897	Tables not assigned to a Databank
18	98.31%	10	590	Tables in META_OBJECTS not present
21	98.53%	2	136	Views in META_OBJECTS not present
26	16.67%	125	150	Views that are incorrectly named
28	99.83%	1	600	Triggers incorrectly named
30	88.03%	57	476	Indexes incorrectly named
38	91.15%	10	113	History / Audit tables; wrong name
44	93.33%	40	600	Triggers not documented
50	98.05%	8	411	Primary Key (PK) constraints wrong name
207	88.58%	73	639	Non-Confid tables without Public Select
211	99.83%	1	596	Unauthorised Alter privileges on tables
501	99.34%	43	6510	Tables without Primary Keys

This is a fabricated example, for illustrative purposes only. The real alerts cover several instances, and cover much more specific detail. Nevertheless, the principle of the data being reported is quite clear.

The overall health of the database (insofar as the term has any validity!) can be expressed by examining the total number of exception tests in association with the total number of exception failures, for the current period. This is actually expressed through a view with the following definition:

```
select
  substr((
    round(100 - ((sum(alert_count)/sum(alert_ref))*100),2) || '%',1,12) "
DB_HEALTH",
  sum(alert_count) total_errors,
  sum(alert_ref) total_measured
from
  alerts_log_health
where
  alert# < (select max(alert#) from alerts_log) ;
```

The database health figure is really just a qualitative measure – too much should not be read into it, but it does nevertheless indicate the extent to which exceptions still remain in the database. A typical result from this query might be as follows:

```
DB_HEALTH      TOTAL_ERRORS  TOTAL_MEASURED
-----
97.26%                4527          165029
```

... which indicates that out of a total population of 165,029 individual conditions examined, 4,527 fail to meet the required standards. Therefore 97.26% of the data is conformant.

9. Conclusions

This paper has described a means whereby corporate metadata can be checked on a regular basis for conformance to the standards laid down by the organisation. Why is this important? Quite sim-

ply, because it is one crucial measure of *Data Quality*. In the extreme case of a company having the veracity of their data challenged in law, it is essential to be able to demonstrate in a coherent, structured and logical manner that data is being managed in a systematic fashion.

The proper application of metrics enables the validation of the data store to be placed firmly into the domain of quantitative rather than qualitative conformance. Every one of the tests used in this work results either a true or false response. No check can possibly return a value of 'perhaps'. In this respect, the principle of two-valued logic is being adopted, which is of course a fundamental characteristic of a properly designed relational data model.

Ultimately, a data model is a representation of a real world situation. For the model to be viewed with confidence as a true representation of that real world situation, it must demonstrate the quantitative conformance to the situation being modeled as far as is ever possible.

The efficient production of these alert reports depends on the alert infrastructure being held in the database. The alert dictionary is the key component, as it functions as the 'driver' for the storage of alert counts, the dissemination of all the e-mail warnings, and the generation of the intranet alert report pages. The only components that have to be manually coded are the two SQL scripts, to generate the actual alert query, and the count of the number of alert conditions. Once these are built, and the dictionary populated, the daily 'cron' job will automatically generate all the relevant reports and e-mails.

Bibliography

- [Rob00] Robson,P.G.: Technical Metadata in the British Geological Survey. PLOUG2000, Zakopane.