Oraculum

Implementing various Oracle practices

Vladimir Ritz Bossicard (vbossica@gmail.com)

The main problem with best practices is that you always discover them too late. In the Java world, we enjoy refactoring tools but I haven't found anything similar for PL/SQL scripts. Since it's not quite easy to change direction, you better start right.

This project regroups various interesting and hopefully best practices for developing with Oracle. I am particularly interested in seeing how these practices could be implemented in my personal work flow.

Managing PL/SQL Exceptions

PL/SQL features an exception mechanism and there is no reason not to take advantage of it. The only problem is that exceptions are buried deep into the code bowels and it's quite difficult to keep an oversight over what exception were already defined and what numbers are already taken. Since Steven Feuerstein recommends in his *Oracle PL/SQL Programming* [http://oreilly.com/catalog/9780596009779/] to manage the exception values into a database table, let's follow his advice. The table only differs with the addition of the `msgmodule` column, which purpose will be described later.

Figure 1. msginfo table

```
CREATE TABLE msg_info (
    msgcode INTEGER,
    msgtype VARCHAR2(30),
    msgmodule VARCHAR2(30),
    msgtext VARCHAR2(2000),
    msgname VARCHAR2(30),
    description VARCHAR2(2000)
);
```

I guess I will never be a true DBA because when it comes to generating files, I always go back to my Java toolkit. Anyway, here's the resulting `errnums.pks` package file...

Figure 2. errnums PS/SQL package

```
CREATE OR REPLACE PACKAGE errnums
IS

    exc_emp_too_young EXCEPTION;
    en_emp_too_young CONSTANT INTEGER := -20200;
    PRAGMA EXCEPTION_INIT (exc_emp_too_young, -20200);

    exc_bal_too_low EXCEPTION;
    en_bal_too_low CONSTANT INTEGER := -20100;
    PRAGMA EXCEPTION_INIT (exc_bal_too_low, -20100);

END errnums;
```

generated by an Ant target so that this step can easily be included into an automated process (e.g. nightly build).

Figure 3. msginfo Ant task

```
<msginfo url="jdbc:oracle:thin:@sausalito:1521:XE" username="scott" password="tiger"
    template="errnums.pks.ftl" outFile="errnums.pks"/>
```

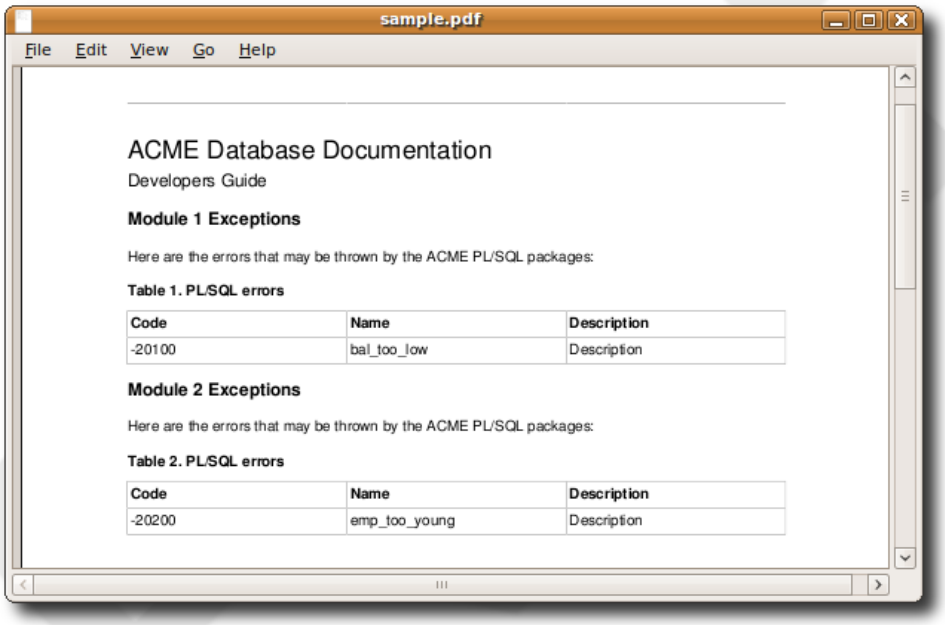
One of the advantages of working with templates is that it is trivial to generate other output files and since no project should come without documentation, let's generate -using the next Ant target- some Docbook fragment and include them (automatically) into the database documentation:

Figure 4. msginfo Ant task for Docbook

```
<msginfo url="jdbc:oracle:thin:@sausalito:1521:XE" username="scott" password="tiger"
    template="errnums.dbk.ftl" outFile="errnums.docbook"/>
```

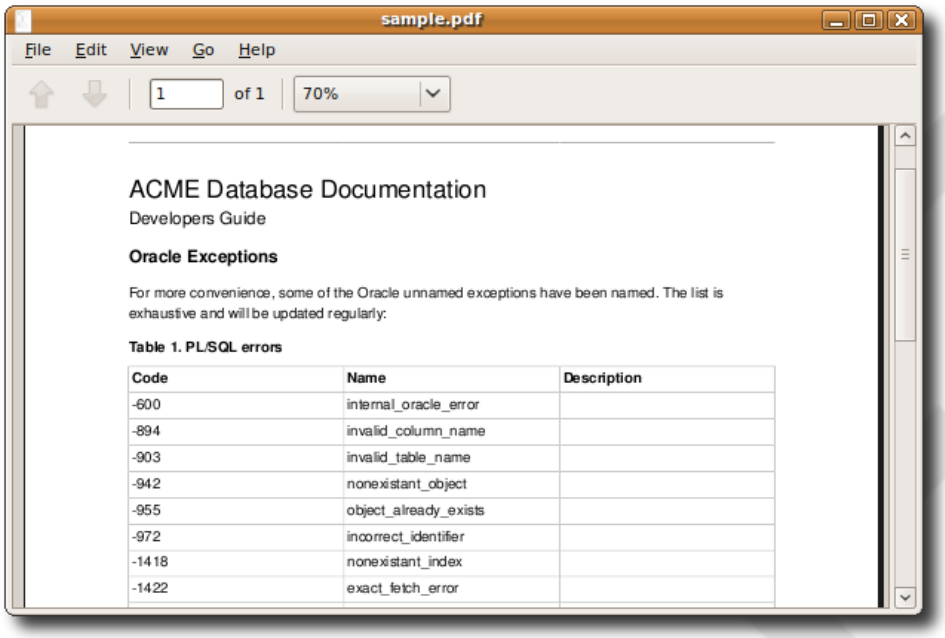
You will notice that the exceptions have been grouped by modules but they can as easily be listed together and included into an appendix.

Figure 5. Application's exceptions documentation



Now that we have our specific exceptions documented, it is trivial to follow the same process and document some of Oracle's numerous unnamed exceptions. These exceptions will automatically be included into the application `errnums.pks` package and also the database guide:

Figure 6. Oracle's unnamed exceptions documentation



Data Dictionary

Having a database is good but it is unfortunately often not straightforward to understand all the intricacies of a rapidly evolving design. Oracle offers the possibility to add comments to tables and columns but leaves the developer with SQL*Plus to search for the appropriate information. One way to solve this problem is to write a *Data Dictionary* document that presents not only the tables but also explains in details their relations. In this project, we will extract the comments from the database itself and generate *Docbook* [http://docbook.org/] XML fragments so that they can be incorporated into the document.

Defining comments for the tables and columns is as simple as invoking the Oracle `comment on` procedure. Please note that the comments are formatted with the *MediaWiki* [http://mediawiki.org/] language.

Figure 7. Oracle comments

```
comment on table msg_info is 'MSG_INFO stores the list of Oracle mappings.';

comment on column msg_info.msgmodule is
'Name of the module (or packages group) declaring this exception. The possible values are:
* Oracle
* any other application name';
comment on column msg_info.msgtext is 'Short text used as describing the exception';
```

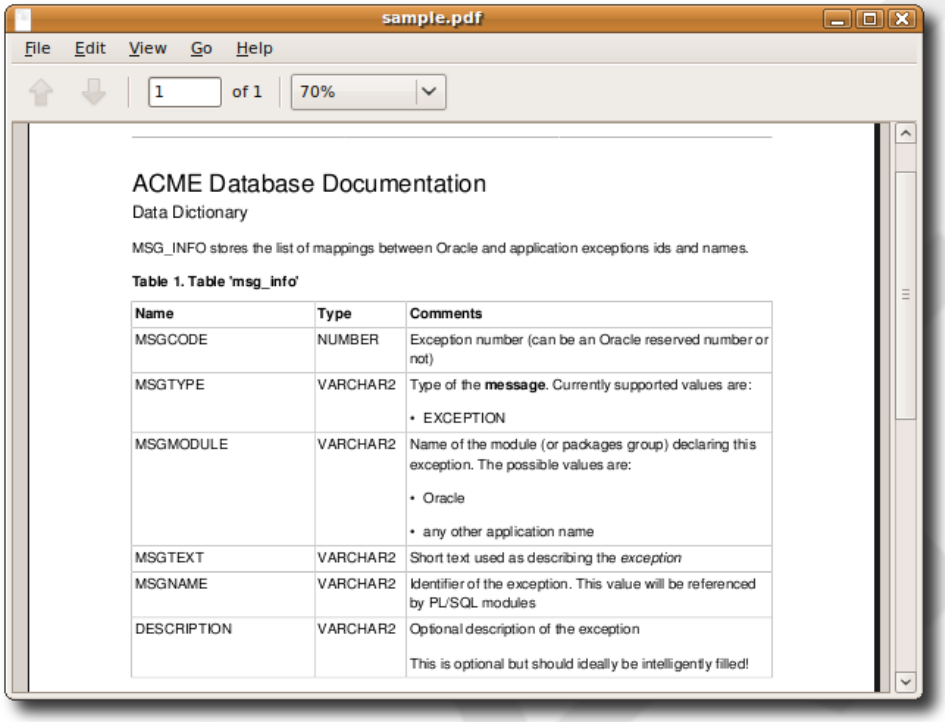
Once the comments have been entered, the only thing left to do is to use the `tableinfo` Ant target to generate the documentation for a given table:

Figure 8. Tableinfo Ant task

```
<tableinfo url="jdbc:oracle:thin:@sausalito:1521:XE"
    username="scott" password="tiger"
    table="msg_info" outFile="${target.dir}/test/msg_info.docbook"/>
```

Using the MediaWiki language for the comments allow us to format them nicely when they are rendered into the final document. This is very handy to present lists, multiple paragraphs or emphasis a term. As is it the case with Docbook, it is very easy to mix and match several sources to produce the final *Data Dictionary* document:

Figure 9. Generated data dictionary document



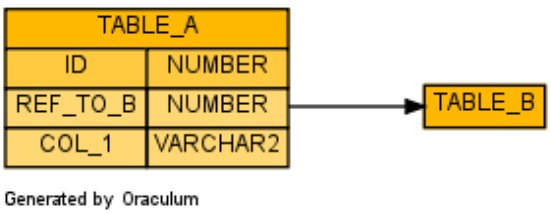
Automatically listing the fields of a table is a good way to describe a table's content, but using a graph to depict the various relationship between the tables is sometimes more useful. An Ant task reads the metadata from the database and produces a `dot` file that can then be rendered by Graphviz:

Figure 10. tablegraph Ant task

```
<tablegraph url="jdbc:oracle:thin:@sausalito:1521:XE"
    username="scott" password="tiger"
    tables="msg_info" outFile="msg_info.dot"/>
```

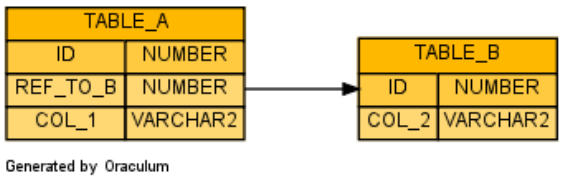
The first example shows how a single table being graphed. Note that the first level relationships with other tables are automatically depicted:

Figure 11. Single table schema



Of course it is possible to graph several tables simultaneously. This time any relationship existing between them is graphed with their respective fields.

Figure 12. Multiple tables schema



Of course an Oracle backed application doesn't only use tables but also PL/SQL modules and it can be useful to see the relationship between those PL/SQL modules.

Figure 13. tablegraph Ant task

```
<pkggraph url="jdbc:oracle:thin:@sausalito:1521:XE"
    username="scott" password="tiger"
    package="core_package" owner="SCOTT" outFile="msg_info.dot"/>
```

will display the package's dependencies as follow:

Figure 14. Package's dependencies