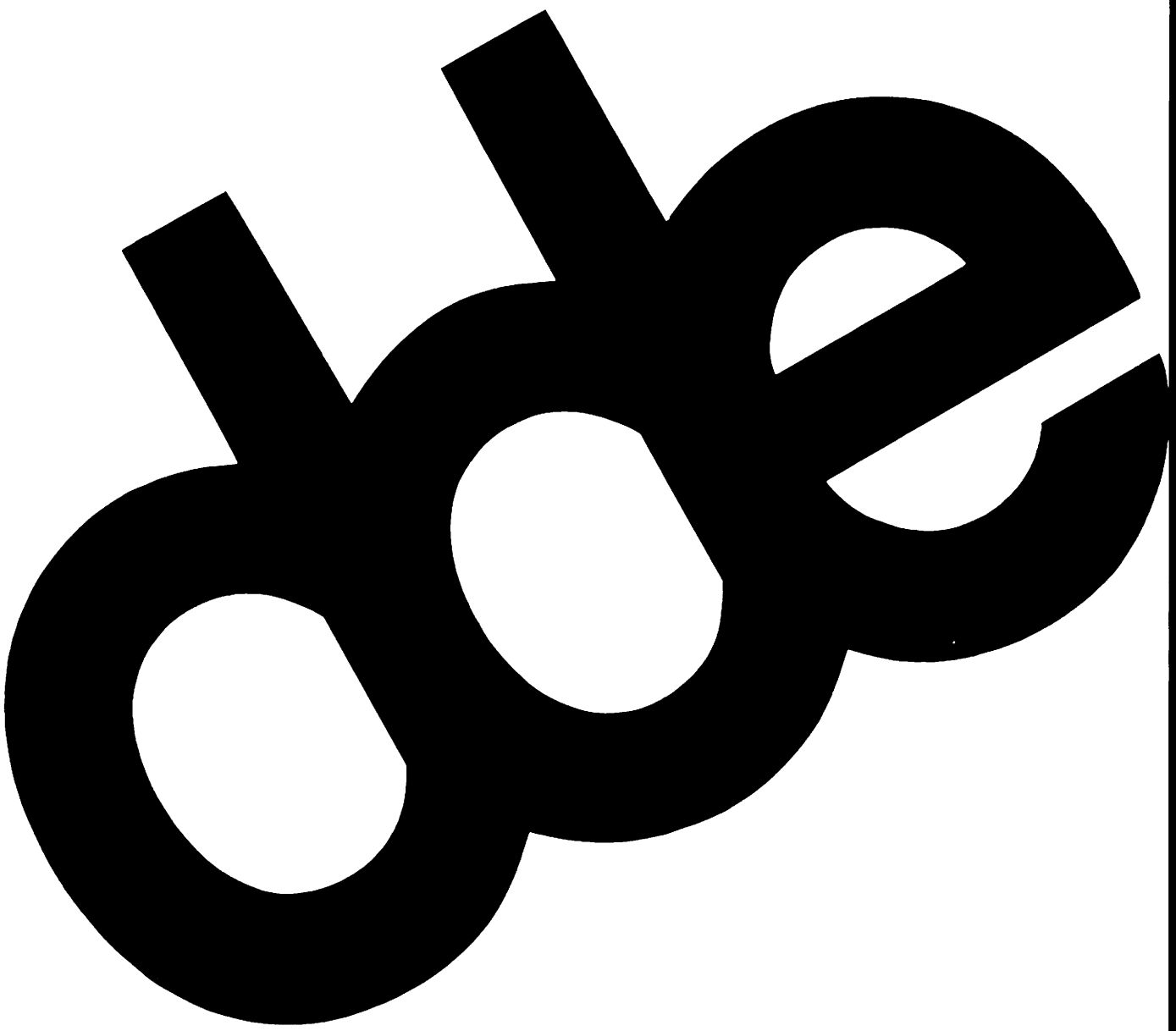
Designer/2000 Wizards



Dedicated Wizards

2. July 1996
by Martin Jensen
Toftekærsvej 113, DK 2860 Søborg
Denmark
tlf: (+45) 39 66 03 13

This presentation will show how the Designer/2000 product from Oracle may be adjusted to fit the style and discipline of a given project, rather than changing the project to fit the rules of the tool.

The concept and scripts where developed by Martin Jensen as an employee at Dansk Data Elektronik A/S - Denmark in March this year.

Designer/2000 Concepts

Obviously a complex tool like Designer/2000 includes a number of rules and assumptions on how the tool should help the analysts, designers and implementers doing their jobs in an appropriate way. Most of these rules and assumptions are based on best practice and common sense.

Normally a project should try to adopt the assumptions and rules explicitly (or implicitly) used by the tool in order to get most value from using the tool. But occasionally, a project has another focus, or should deal with issues which could not be foreseen by any tool vendor.

Why should we reject a good tool just because we need some additional rules inside Designer/2000?

Some of the Rules

Foreign key specifications should not be given at the analyse stage. The use of relationships should be used to let the *Data Design Wizard* create the appropriate foreign key columns at the design level with a given naming convention.

The generators will produce code with a certain look. E.g. the DDL generator will not precede any *create* statement with a *drop* statement, ...

A given object naming convention is difficult to enforce by the tool. And it is hard to demand that domain specifications should be mandatory for all attributes and columns.

And maybe the project requires that many levels of domains must be used including a scheme of inheritance for some of the properties.

From a general point of view, it is indeed difficult to let the *Application Design Wizard* modify an existing module from changes in the function hierarchy. On the other hand one may want a number of common cases to be treated in order to avoid rudimentary jobs recreating modules during overtime in weekends.

And maybe the project requires that some sort of tracking is build into the retrofit facility, in order for the model manager to identify where the various information originally came from. A project may also request the ability to trace fx. which areas of the repository did change when.

How can this be achieved

Observe that the designers of Designer/2000 did already build some documented flexibility into the tool.

The data model of the Designer/2000 repository is actually documented as ER-diagrams and in the on-line documentation. API's, a number of PL/SQL procedures on modifying (almost) all objects and associations in the repository are documented. Each object and association may be extended with a number of user defined properties. And even new objects and associations may be defined.

One of objects not covered by documented API procedure is free length text objects stored in the CDI_TEXT table. The columns of this table are:

1. **txt_ref** - The element id of the element being described here, such as the id of a table element.
2. **txt_type** - The kind of text associated with the element - such as CDHELP for the appropriate help text.
3. **txt_seq** - The line number of the line in the text for the association to the element.
4. **txt_text** - The actual line of text.

A Dedicated Wizard

A *Dedicated Wizard* is a program that enforces a certain rule or discipline among objects in the repository. Assume we want to implement a *Wizard* for inheriting given properties through multiple levels of domains.

First the *Domain* (DOM) element type is selected through the Management option, and one of the unused user extensible properties are selected - say number 7. This property could be labelled *super_domain_name*, and take up a maximum of 30 characters. (Note that this will make it easy for the operator to view and enter super domain names, but will make it difficult to change the actual domains, since Designer/2000 uses numbers to identify elements internally)

Then a SQL*Plus PL/SQL script using the Designer/2000 API may be developed. The script assumes that all domain elements in the hierarchy is owned by the same application.

A Wizard Script - for SQL*Plus

```
set verify off
declare
   dom                  ciodomain.data;
   l_dom_id             ci_domains.id%type;
   l_dom_name           ci_domains.name%type;
   l_super_dom_name     ci_domains.name%type;
   l_app_sys_no         ci_domains.application_system_owned_by%type;
   l_app_sys            ci_application_systems.name%type;
   l_rep_change         Varchar2( 1 );      /* may repository be changed? */
   l_lineno             Number;             /* current line # in report */
   l_changed            Number;             /* # of changed objects */
   act_status           Varchar2( 1 );
   act_warnings         Varchar2( 1 );
   rec_changed          Varchar2( 1 );      /* has domain been changed? */
```

```
-- Cursor to get the latest version of an application system:
   cursor get_app_sys_id is
      select id, version, version_date from ci_application_systems
      where name = l_app_sys
       and version = ( select max( version ) from ci_application_systems
                       where name = l_app_sys );
   app_rec                    get_app_sys_id%rowtype;
--
-- Cursor to get a certain super domain:
   cursor get_d2k_domain is
      select * from ci_domains
      where application_system_owned_by = l_app_sys_no
       and name =  l_super_dom_name;
   sup_dom_rec                get_d2k_domain%rowtype;
```

```
-- Cursor to scan the domains in an appropriate order:
   cursor scan_d2k_domains is
      select 1 lev, d.*, s.name super_domain_name
      from ci_domains d, ci_domains d2, ci_domains s
      where d.application_system_owned_by = l_app_sys_no
       and d2.application_system_owned_by = l_app_sys_no
       and d.supertype_reference = s.id( + )
       and d.user_defined_property_7 = d2.name
       and d2.user_defined_property_7 is null
      union
      select 2 lev, d.*, s.name super_domain_name
      from ci_domains d, ci_domains d2, ci_domains d3, ci_domains s
      where d.application_system_owned_by = l_app_sys_no
       and d2.application_system_owned_by = l_app_sys_no
       and d3.application_system_owned_by = l_app_sys_no
       and d.supertype_reference = s.id( + )
       and d.user_defined_property_7 = d2.name
       and d2.user_defined_property_7 = d3.name
       and d3.user_defined_property_7 is null
      union /* ... for as many levels as you like ... */
```

```
-- Procedure to write lines - implemented through an RDBMS table:
   procedure write_line( t Varchar2 ) is
   begin
      insert into wizard_report ( line_no, line_text ) values ( l_lineno, t );
      l_lineno := l_lineno + 1;
   end write_line;
--
   procedure instantiate_messages is
-- Taken from the Designer/2000 API example in the on-line documentation.
```

```
-- The main program:
begin
   delete from wizard_report; commit;
   l_rep_change := upper( '&1' );
   l_app_sys := upper( '&2' );
   l_lineno := 1;
   l_changed := 0;
--
-- Welcome:
   write_line( 'Wizard on Designer/2000 for Domain Hierarchies on Application system '||
            l_app_sys ||', date: '|| to_char( sysdate,'dd-mm-yyyy hh24:mi:ss')||' by '|| user );
   write_line( 'The Repository change option is set to: '|| l_rep_change );
--
-- First get the appropriate application id:
   open get_app_sys_id;
   fetch get_app_sys_id into app_rec;
   if get_app_sys_id%notfound then
      write_line( 'Application system '|| l_app_sys ||' could not be found!' ); return;
   else
      l_app_sys_no := app_rec.id;
      write_line( 'Application system '|| l_app_sys ||' of version '|| to_char( app_rec.version )||
               ' from '|| to_char( app_rec.version_date,'dd-mm-yyyy hh24:mi:ss' )||
               ' has id: '|| to_char( l_app_sys_no ));
   end if;
   close get_app_sys_id;
--
-- dbms_output.enable( 32000 ); -- From the time when I tried to use dbms_output.put_line!
   cdapi.initialize( l_app_sys );
```

```
-- Now, let us loop through all the interesting domains:
   for dom_rec in scan_d2k_domains loop
      cdapi.open_activity;
      l_dom_id := dom_rec.id;
      l_dom_name := dom_rec.name;
      l_super_dom_name := upper( dom_rec.user_defined_property_7 );
--
-- Check if the supertype_reference is used, then it is consistent with UE7
      if ( dom_rec.super_domain_name is not null ) and
         ( dom_rec.super_domain_name != l_super_dom_name ) then
         write_line( 'Error in domain '|| l_dom_name ||' has super domain '|| l_super_dom_name ||
                     ' but references domain '|| dom_rec.super_domain_name );
      end if;
--
-- Let us check that the specified super domain does actually exist:
      open get_d2k_domain;
      fetch get_d2k_domain into sup_dom_rec;
      if get_d2k_domain%notfound then
         write_line( 'Error, super domain '|| l_super_dom_name ||
                     ' does not exist for domain '|| l_dom_name );
      else
```

```
-- Get the actual domain into the working area:
        ciodomain.sel( l_dom_id, dom );
        rec_changed := 'N';  /* no changes to the domain yet! */
--
-- Inherit the data-type property:
        if ( nvl( dom.v.datatype, 'null' ) != nvl( sup_dom_rec.datatype, 'null' )) then
           rec_changed := 'Y';
           write_line( '  datatype := '|| sup_dom_rec.datatype ||' from '|| dom.v.datatype );
           dom.v.datatype := sup_dom_rec.datatype;
           dom.i.datatype := true;
        end if;
-- Inherit other properties like precision, maximum_length, derivation, format, ... if you want so.
--
-- Update the  object in the repository if necessary:
        if rec_changed = 'Y' then
           l_changed := l_changed + 1;
           write_line( 'Designer/2000 last changed date: '||
                    to_char( dom.v.data_changed,'dd-mm-yyyy hh24:mi:ss' ));
           if l_rep_change = 'Y' then
              ciodomain.upd( l_dom_id, dom );
              write_line( 'Domain '|| l_dom_name ||' updated' );
           end if;
        end if;
     end if;
```

```
-- Now, clean up and prepare for the next loop
      close get_d2k_domain;
      cdapi.validate_activity( act_status, act_warnings );
      instantiate_messages;
      if act_status != 'Y' then
         cdapi.abort_activity;
         write_line( 'Activity aborted with constraint violations!' );
      end if;
      commit;
   end loop;
   close scan_d2k_domains;
--
-- Report some statistics:
   if l_rep_change = 'Y' then
      write_line( 'Domains updated: '|| to_char( l_changed ));
   else
      write_line( 'Domains which could have been updated: '|| to_char( l_changed ));
   end if;
   write_line( 'Wizard on Designer/2000 on Domain hierarchies Terminated.' );
exception
   when others then
      if cdapi.stacksize > 0 then
--       Taken from the API example ....
end;
/
```

```
rem The Wizard Report is now returned (assuming SQL*Plus is used):
set heading off
set arraysize 1
set feedback off
set pagesize 0
select line_text from wizard_report order by line_no;
exit
```

Good luck and best wishes
Martin Jensen

Dedicated Designer/2000 Wizards

This document is a detailed description on which information is maintained from the Designer/2000 Dedicated Wizard. The document also tries to guide the developer through some of the activities to be done on some of the stages.

1. Products and Versions

These scripts have been verified to run with the following tools:

Oracle7 rdbms	Oracle7 Server Release 7.1.4.1.0
PL/SQL	PL/SQL Release 2.1.4.0.0
SQL*Plus	SQL*Plus: Release 3.1.3.5.1
Designer/2000 1.1	1.0.7

2. General Concept

The general function of the wizards are to help development with moving appropriate information from one level to the other.

3. Install Designer/2000

Use the *Admin Utility - User Extensibility* to map the used USER_DEFINED_PROPERTY_0 - 9 properties to meaningfull names, and make them appear in the Repository Object Navigater (RON).

3.1 Domain User Extensions

PROPERTY_7 is the mother_domain name of length 30.

3.2 Column User Extensions

PROPERTY_0 is the export_time_text from other case tool, called *export_time_text* of length 80.

PROPERTY_1 is the long_name of the object if it exists, called *long_name* of length 80.

PROPERTY_2 is the format_mask of the object if it exists, called *format_mask* of length 80.

PROPERTY_3 is the security_level of the object, called *security_level* of length 80.

PROPERTY_4 is the history of the object, called *history* of length 200.

PROPERTY_5 is the prompt of the object, called *prompt* of length 50.

PROPERTY_6 is the uppercase constraint of the object, called *uppercase* of length 80.

PROPERTY_7 is the composed attribute list of the object, called *composed_list* of length 100.

PROPERTY_9 is the accept_code of the object, called *accept_code* of length 10.

3.3 Table User Extensions

PROPERTY_0 is the export_time_text from other case tool, called *export_time_text* of length 80.

PROPERTY_1 is the long_name of the object if it exists, called *long_name* of length 80.

PROPERTY_3 is the security_level of the object, called *security_level* of length 80.

PROPERTY_4 is the history of the object, called *history* of length 200.

PROPERTY_7 is the revision, called *revision_code* of length 10.

PROPERTY_9 is the accept_code of the object, called *accept_code* of length 10.

3.4 Sequence User Extensions

PROPERTY_0 is the export_time_text from other case tool, called *export_time_text* of length 80.

PROPERTY_3 is the security_level of the object, called *security_level* of length 80.

PROPERTY_4 is the history of the object, called *history* of length 200.

PROPERTY_9 is the accept_code of the object, called *accept_code* of length 10.

3.5 Primary Key Constraint User Extensions

PROPERTY_0 is the export_time_text from other case tool, called *export_time_text* of length 80.

PROPERTY_1 is the long_name of the object if it exists, called *long_name* of length 80.

PROPERTY_3 is the security_level of the object, called *security_level* of length 80.

PROPERTY_4 is the history of the object, called *history* of length 200.

PROPERTY_9 is the accept_code of the object, called *accept_code* of length 10.

3.6 Foreign Key Constraint User Extensions

PROPERTY_0 is the export_time_text from other case tool, called *export_time_text* of length 80.

PROPERTY_1 is the long_name of the object if it exists, called *long_name* of length 80.

PROPERTY_3 is the security_level of the object, called *security_level* of length 80.

PROPERTY_4 is the history of the object, called *history* of length 200.

PROPERTY_5 is the method how to change the child side of the relation when the parent chang es - is called *change_method* of length 80.

PROPERTY_9 is the accept_code of the object, called *accept_code* of length 10.

3.7 Key Component User Extensions

PROPERTY_0 is the export_time_text from other case tool, called *export_time_text* of length 80.

PROPERTY_1 is the long_name of the object if it exists, called *long_name* of length 80.

PROPERTY_3 is the security_level of the object, called *security_level* of length 80.

PROPERTY_4 is the history of the object, called *history* of length 200.

PROPERTY_9 is the accept_code of the object, called *accept_code* of length 10.

3.8 Index User Extensions

PROPERTY_0 is the export_time_text from other case tool, called *export_time_text* of length 80.

PROPERTY_1 is the long_name of the object if it exists, called *long_name* of length 80.

PROPERTY_3 is the security_level of the object, called *security_level* of length 80.

PROPERTY_4 is the history of the object, called *history* of length 200.

PROPERTY_9 is the accept_code of the object, called *accept_code* of length 10.

3.9 Module Detail Table Usage User Extensions

PROPERTY_0 is the export_time_text from other case tool, called *export_time_text* of length 80.

PROPERTY_1 is the long_name of the object if it exists, called *long_name* of length 80.

PROPERTY_3 is the security_level of the object, called *security_level* of length 80.

PROPERTY_4 is the history of the object, called *history* of length 200.

PROPERTY_7 is the revision, called *revision_code* of length 10.

PROPERTY_9 is the accept_code of the object, called *accept_code* of length 10.

3.10 Module Detail Column Usages User Extensions

PROPERTY_0 is the export_time_text from other case tool, called *export_time_text* of length 80.

PROPERTY_1 is the long_name of the object if it exists, called *long_name* of length 80.

PROPERTY_2 is the format_mask of the object if it exists, called *format_mask* of length 80.

PROPERTY_3 is the security_level of the object, called *security_level* of length 80.

PROPERTY_4 is the history of the object, called *history* of length 200.

PROPERTY_5 is the prompt of the object, called *prompt* of length 50.

PROPERTY_6 is the uppercase constraint of the object, called *uppercase* of length 80.

PROPERTY_7 is the composed attribute list of the object, called *composed_list* of length 100.

PROPERTY_9 is the accept_code of the object, called *accept_code* of length 10.

4. Run the Dedicated Wizards

The scripts *wizard.sh* and *wizard_mod.sh* will do the job for you. The parameter should be the name of the application in Designer/2000, and the environment D2KREP_PASS should be set to the Oracle account holding the Designer/2000 repository tables. And the repository owner must have a priviledge to update (and maybe delete from the application).

Prior to the run the Database Design Wizard from Designer/2000 would have to be executed, in order to populate all relevant objects at the design level. And you are free to choose which entity objects you want the Database Design Wizard to deal with. The Dedicated Wizards will try to extend the information in the design objects already there, but will not create new table objects.

The scripts will not try to insert any new major objects in the Designer/2000. If an object is there however, it will check if some of the porperties are to be initiated or changed.

The scripts will **not** try to delete any extra objects in the Designer/2000 repository not pressent in the analyze level, but will only inform the operator of their existence. The scripts will thus try to be tollerent, and may be executed to update the Designer/2000 repository design level with changes from the analyze level.

One report is generated by each of the wizards. And as the report is generated by inserting all the lines on a table, which are selected afterwards, these scripts may only be executed one at the time. The table holding the report is called **wizard_report** and is created with the SQL*Plus script called *wizard_1.sql.*

4.1 The Data-side Wizard

The file wizard_change.$APP_SYS.txt is a report of the actual transport of objects. Which existing objects are updated with changed values, and which objects exists in the design and not on the analyze level of Designer/2000 repository, Generated by *wizard.sh.*

4.2 The Application-side Wizard

The file wizard_mod_chg.$APP_SYS.txt is a report of the actual transport of objects. Which existing objects are updated with changed values, and which objects exists in the design and not on the analyze level of Designer/2000 repository, Generated by *wizard_mod.sh.*

4.3 The Entity Table Mapping Wizard Part

Before running the Database Design Wizard from Designer/2000 it is sometimes required to insert mapping information to force the wizard to map existing entities on to predefined tables. This is especial relevant when information is imported from other tools where this mapping information exists.

This wizard part *d2k_ent_map.sql* assumes that the table name has been placed in the User_Defined_property_1 property, and will create table as well as mapping information, if not already there.

4.4 The Module Table Usage Wizard

A special Wizard has been made to take a list of Modul names, table names and their usage and load this information first to a new repository table (**dde_module_usages**), and then to update the repository with this information.

The file takes the following format: For each line:

<Module_name> <Table_name> <sel_flag> <ins_flag> <upd_flag> <del_flag>
as:
M00100 EMP OBL OBL 0 0

Where a flag 'OBL' meens mandatory, and '0' meens not used. Observe that also view names may be given as <Table_name>.

The *$DEVENV/D2K/mod_usg_lo.sh* script will load the information from the file into the **dde_module_usages**) table. And the *$DEVENV/D2K/d2k_mod_usg.sql* SQL*Plus script will update the repository with this information.

4.5 The Super Domain Hierarcky Wizard

Designer/2000 does not support more than 2 levels of domains, and does not support inheritance from super to subdomain. This wizard is based on the user defined property 7 (*mother_domain* name) and will try to let every sub domain inherit properties from the superdomain starting from the top.

The limitations are that only 4 levels are supported - and that domain names entered are not automatically changed, when the actual domain object does change name.

The following properties are inherited: Datatype, attribute_precision, column_precision, derivation, format, maximum_attribute_length, maximum_column_length, format_mask (UE2), security_level (UE3), Dispaly_length (UE5), uppercase (UE6).

Remember to activate *Update Attributes in a Domain*, and *Update Columns in a Domain* afterwards.

4.6 Examples

```
# prepare for the data-side wizard by inserting relevant table and mapping information.
$ sqlplus -s $D2KREP_PASS @$DEVENV/D2K/d2k_ent_map Y MY_APPLICATION
# Now run the Database Design Wizard from Designer/2000 !
# Run the data-side wizard.
$ $DEVENV/D2K/wizard.sh MY_APPLICATION N D
# Run the application-side wizard.
$ $DEVENV/D2K/wizard_mod.sh MY_APPLICATION N
#
# Or if you just want to clear the candidate property of
# all generated modules - you may run:
$ sqlplus -s $D2KREP_PASS @$DEVENV/D2K/d2k_mod Y MY_APPLICATION
#
# Load a set of Module Table Usages into the repository:
# First fill the dde_module_usage table - and then update the repository.
$ D2KREP_PASS= ... $DEVENV/D2K/mod_usg_lo.sh mod_usg.txt first
$ sqlplus -s $D2KREP_PASS @$DEVENV/D2K/d2k_mod_usg Y MY_APPLICATION
#
# Let all domains inherit properties from their super domains.
$ sqlplus -s $D2KREP_PASS @$DEVENV/D2K/d2k_dom2 Y MY_APPLICATION
```

Note. Add a parameter *N* to the command line if the wizard should not update the repository, in which case only the *pre* log file is generated.

The *wizard.sh* and *wizard_mod.sh* scripts consists of a number of PL/SQL scripts to to the actual transfer of the different types of objects.

4.6.1 Stand-alone Wizards Keeping a lot of information in the repository updated can be quite time consuming. the following small scripts will help you. Again option *Y* updates the repository, and *N* does not but generates a report.

If you just want to clear the *candidate* property of all generated modules - you may run:

```
$ sqlplus -s $D2KREP_PASS @$DEVENV/D2K/d2k_mod option application_name
```

Or if you just want to clear the *Display_format* and *display_length* properties of all module detailed column usages (DCU) refering to *date, timestamp* or *time* columns, as preferences for this exists - you may run:

```
$ sqlplus -s $D2KREP_PASS @$DEVENV/D2K/d2k_dcu_upd option application_name
```

4.7 Moving Entities to Table Definitions

The *d2k_tab.sql* script will do the job. For all rows in the ci_entities view an entity object is checked against the Designer/2000 repository. And the following attributes in the Designer/2000 repository are set, and some of the properties are set even if no corresponding entity exists, marked with *(tab)*.

ALL_DATABASES_IND is set to 'Y', if no explicit database is set. *(tab)*

DISPLAY_TITLE is set to same as Initcap of the table name. *(tab)*

JOURNAL_FLAG is set to 'Y' if the security level is 3 or 4, otherwise set to 'N'. *(tab)*

REMARK is set to the value of long_name. And if *long_name* is null, the first 120 characters from the description is taken if any exists. *(tab)*

USER_DEFINED_PROPERTY_0 is set to the USER_DEFINED_PROPERTY_0 value from the entity.

USER_DEFINED_PROPERTY_1 is set to the USER_DEFINED_PROPERTY_1 value from the entity.

USER_DEFINED_PROPERTY_3 is set to the USER_DEFINED_PROPERTY_3 value from the entity.

USER_DEFINED_PROPERTY_4 is set to the USER_DEFINED_PROPERTY_4 value from the entity.

USER_DEFINED_PROPERTY_7 is set to the USER_DEFINED_PROPERTY_7 value from the entity.

USER_DEFINED_PROPERTY_9 is set to the USER_DEFINED_PROPERTY_9 value from the entity.

4.8 Moving KEY Attribute information to Sequences

The *d2k_seq.sql* script will do the job. For all rows in the ci_attributes view using the KEY domain, a sequence object is checked against the Designer/2000 repository. And the following attributes in the Designer/2000 repository are set:

NAME is the same as the attribute name.

USER_DEFINED_PROPERTY_0 is set to same value as the attribute USER_DEFINED_PROPERTY_0 value.

USER_DEFINED_PROPERTY_3 is set to same value as the attribute USER_DEFINED_PROPERTY_3 value.

USER_DEFINED_PROPERTY_4 is set to same value as the attribute USER_DEFINED_PROPERTY_4 value.

USER_DEFINED_PROPERTY_9 is set to same value as the attribute USER_DEFINED_PROPERTY_9 value.

4.9 Moving Attribute information to Columns

The *d2k_col.sql* script will do the job. For all rows in the ci_attributes view a column object is checked against the Designer/2000 repository. And the following attributes in the Designer/2000 repository are set:

Note that the Designer/2000 Database Design Wizard will create columns used for foreign keys. But as they are taken into account from the model we are moving from Analyze to Design these extra columns are to be removed again! BTW: No columns are removed if the option *O* is used instead of option *Y*!

The Designer/2000 Database Design Wizard will also create extra columns when sub entities are included in the table mapping a super entity. Such columns should not be deleted by the dedicated wizard.

So this script will remove all columns in the design level not mentioned at the analyze level, and having the User defined History property as null, and with no link to *Table Entity Usages*. It is thus still possible for the designer to add extra columns in at this level as long as the history property are set to anything but null, and some of the properties are set even if no corresponding attribute exists, marked with *(col)*.

AUTO_GENERATED	if the domain derivation property is *Auto_Gen: Created by* this property is set to *UC*, *Auto_Gen: Modified by* this property is set to *UM*, *Auto_Gen: Date Created* this property is set to *DC*, *Auto_Gen: Date Modified* this property is set to *DM*, *Auto_Gen: Seq in Parent* this property is set to *SP*. The match is not case sensitive.*(col)*
UPPERCASE	is first set from the domain uppercase field if that value exist, otherwise set from the attribute uppercase field. The format is assumed to be *Uppercase: T|F*, where *T* is translated to *Y* and otherwise *N.(col)*
DISPLAY_LENGTH	the length of format - If it exists, plus one for numerical types to allow for a sign. If the format mask not exists and the datatype is *Date* display length is set to 10 (to allow for dd/mm-yyyy). If the datatype is numeric and the Designer/2000 Wizard did set the display length to 39, the display length is adjusted to 10.*(col)*
FORMAT_MODIFIER	the actual format mask from the domain. The format is either *'Output Format mask: <format>', Input Format mask: <format>*, or simply the actual format mask to be used.*(col)*
PROMPT	the prompt field from the Attribute object.
HELP_TEXT	the long_name field from the Attribute object.

NULL_INDICATOR the optional_flag field from the Attribute object.

ORDER_SEQUENCE is the *order_sequence* from the attribute Object.

REMARK is set to the value of long_name. And if *USER_DEFINED_PROPERTY_1* is null, the first 220 characters from the description is taken if any exists. *(col)*

SEQUENCE_NUMBERis the *sequence_number* from the attribute Object.

SEQUENCE_REFERENCE is set to the proper sequence id if it is specified that this column should rely on a sequence.

USER_DEFINED_PROPERTY_0 is set to same value as the attribute USER_DEFINED_PROPERTY_0 value.

USER_DEFINED_PROPERTY_1 is set to same value as the attribute USER_DEFINED_PROPERTY_1 value.

USER_DEFINED_PROPERTY_2 is set to same value as the domain USER_DEFINED_PROPERTY_2 value.

USER_DEFINED_PROPERTY_3 is set to same value as the attribute USER_DEFINED_PROPERTY_3 value.

USER_DEFINED_PROPERTY_4 is set to same value as the attribute USER_DEFINED_PROPERTY_4 value.

USER_DEFINED_PROPERTY_5 is set to same value as the attribute USER_DEFINED_PROPERTY_5 value.

USER_DEFINED_PROPERTY_6 is set to same value as the attribute USER_DEFINED_PROPERTY_6 value.

USER_DEFINED_PROPERTY_7 is set to same value as the attribute USER_DEFINED_PROPERTY_7 value.

USER_DEFINED_PROPERTY_9 is set to same value as the attribute USER_DEFINED_PROPERTY_9 value.

4.10 Moving Unique Identifier information to Primary Key Constraints

The *d2k_pk.sql* script will do the job. For all rows in the ci_unique_identifier_entries view a Primary Key Component object is checked against the Designer/2000 repository. And the following attributes in the Designer/2000 repository are set:

Note that the Designer/2000 Database Design Wizard will create a Primary Key Constraint for every table where a Unique Key constraint exists for the corresponding Entity.

ERROR_MESSAGE is set to *'Primary key on table '||<table_name>||' violated!'*.

IMPLEMENTATION_LEVEL is set to *BOTH*, to reflect that Primary key constraints are to be implemented both on the client and server side.

KEY_UPDATEABLE is set to the value of the relation *Transfer*.

USER_DEFINED_PROPERTY_0 is set to same value as the identifier USER_DEFINED_PROPERTY_0 value.

USER_DEFINED_PROPERTY_1 is set to same value as the identifier USER_DEFINED_PROPERTY_1 value.

USER_DEFINED_PROPERTY_3 is set to same value as the identifier USER_DEFINED_PROPERTY_3 value.

USER_DEFINED_PROPERTY_4 is set to same value as the identifier USER_DEFINED_PROPERTY_4 value.

USER_DEFINED_PROPERTY_9 is set to same value as the identifier USER_DEFINED_PROPERTY_9 value.

4.10.1 Updating Key Components

CONSTRAINT_TYPE is set to *PRIMARY*.

USER_DEFINED_PROPERTY_0 is set to same value as the identifier entry USER_DEFINED_PROPERTY_0 value.

USER_DEFINED_PROPERTY_1 is set to same value as the identifier entry USER_DEFINED_PROPERTY_1 value.

USER_DEFINED_PROPERTY_3 is set to same value as the identifier entry USER_DEFINED_PROPERTY_3 value.

USER_DEFINED_PROPERTY_4 is set to same value as the identifier entry USER_DEFINED_PROPERTY_4 value.

USER_DEFINED_PROPERTY_9 is set to same value as the identifier entry USER_DEFINED_PROPERTY_9 value.

4.11 Moving Relation information to Foreign Key Constraints

The *d2k_fk.sql* script will do the job. For all relevant rows in the ci_relationship_ends view a Foreing Key Constraint object is checked against the Designer/2000 repository. And the following attributes in the Designer/2000 repository are set:

Note that the Designer/2000 Database Design Wizard will create a Primary Key Constraint for every table where a Unique Key constraint exists for the corresponding Entity.

ERROR_MESSAGE is set to *'Foreign key on table '||<table_name>||' violated! -'||<remark>*.

IMPLEMENTATION_LEVEL is set to fIBOTH, to reflect that Foreign key constraints are to be implemented both on the client and server side.

FK_TRANSFERABLE is set to the TRANSFER property from the relation.

FK_CASCADE_DELETE is set to correct value based on USER_DEFINED_PROPERTY_5.

FK_CASCADE_UPDATE is set to correct value based on USER_DEFINED_PROPERTY_5.

USER_DEFINED_PROPERTY_0 is set to same value as the identifier USER_DEFINED_PROPERTY_0.

USER_DEFINED_PROPERTY_1 is set to same value as the identifier USER_DEFINED_PROPERTY_1.

USER_DEFINED_PROPERTY_3 is set to same value as the identifier USER_DEFINED_PROPERTY_3.

USER_DEFINED_PROPERTY_4 is set to same value as the identifier USER_DEFINED_PROPERTY_4.

USER_DEFINED_PROPERTY_5 is set to same value as the identifier USER_DEFINED_PROPERTY_5.

USER_DEFINED_PROPERTY_9 is set to same value as the identifier USER_DEFINED_PROPERTY_9.

4.12 Moving Foreign Key Constraints to Indexes

The *d2k_inx.sql* script will do the job. For all relevant rows in the ci_foreign_key_constraints and ci_key_components views, Index and Index Entry objects are checked against the Designer/2000 repository. And the following attributes in the Designer/2000 repository are set:

Note that the Designer/2000 Database Design Wizard will create an Index for every foreign key, but the dedicated wizard will insert more information, and will create proper index entry elements, as those generated by the Designer/2000 wizard are deleted.

REMARK is set to the ERROR_MESSAGE value from the foreign key constraint.

USER_DEFINED_PROPERTY_0 is set to same value as the foreign key constraint USER_DEFINED_PROPERTY_0 value.

USER_DEFINED_PROPERTY_1 is set to same value as the foreign key constraint USER_DEFINED_PROPERTY_1 value.

USER_DEFINED_PROPERTY_3 is set to same value as the foreign key constraint USER_DEFINED_PROPERTY_3 value.

USER_DEFINED_PROPERTY_4 is set to same value as the foreign key constraint USER_DEFINED_PROPERTY_4 value.

USER_DEFINED_PROPERTY_9 is set to same value as the foreign key constraint USER_DEFINED_PROPERTY_9 value.

4.13 Adjusting Module Information

The *d2k_mod.sql* script will do the job. For all rows in the ci_modules view the following properties are checked in the Designer/2000 repository.

STATUS is set to *null* to reflect that all Modules are not only candidates but for real.

4.14 Moving Table Information to Module Detail Table Usage Information

The *d2k_dtu.sql* script will do the job. For all rows in the ci_module_detail_table_usages view an entity object is checked against the Designer/2000 repository. And the following attributes in the Designer/2000 repository are set:

DISPLAY_TITLE is set to the DISPLAY_TITLE of the table.

USER_DEFINED_PROPERTY_0 is set to the USER_DEFINED_PROPERTY_0 value from the table.

USER_DEFINED_PROPERTY_1 is set to the USER_DEFINED_PROPERTY_1 value from the table.

USER_DEFINED_PROPERTY_3 is set to the USER_DEFINED_PROPERTY_3 value from the table.

USER_DEFINED_PROPERTY_4 is set to the USER_DEFINED_PROPERTY_4 value from the table.

USER_DEFINED_PROPERTY_7 is set to the USER_DEFINED_PROPERTY_7 value from the table.

USER_DEFINED_PROPERTY_9 is set to the USER_DEFINED_PROPERTY_9 value from the table.

4.15 Moving Selected Preference Information to Module Detail Table Usages

The *d2k_dtu_prf.sql* script will do the job. For all rows in the ci_module_detail_table_usages view (of usage *BASE*) some preference values is check ed against the Designer/2000 repository.

If a preference value (at the application level or module level) is found to be bigger than the actual one in the Module Detail Table Usage, it will be set to *null* in the Module Detail Table Usage record, to allow the generator to propagate the new property.

The following attributes in the Designer/2000 repository are set:

REQ_PAGE_HEIGHT is compared with the PAGCHT preference under *LAYOUT - PAGE.*

REQ_PAGE_WIDTH is compared with the PAGCWD preference under *LAYOUT - PAGE.*

REQ_VIEW_HEIGHT is compared with the WINCHT preference under *LAYOUT - WINDOW.*

REQ_VIEW_WDIGHT is compared with the WINCWD preference under *LAYOUT - WINDOW.*

4.16 Moving Column information to Module Detail Column Usages

The *d2k_dcu.sql* script will do the job. For all rows in the ci_module_detail_columns_usages view a column object is checked against the Designer/2000 repository. And the following attributes in the Designer/2000 repository are set:

DEFAULT_VALUE the actual DEFAULT_VALUE from the Column object.

DISPLAY_DATATYPE the actual DISPLAY_DATATYPE from the Column object.

DISPLAY_FORMAT the actual FORMAT_MODIFIER from the Column object.

DISPLAY_LENGTH the actual DISPLAY_LENGTH from the Column object.

HINT_TEXT the actual HELP_TEXT from the Column object.

JUSTIFICATION is set to *RIGHT* if the DISPLAY_DATATYPE from the Column is numeric.

NULLIFY_FLAG is set to *Y* if NULL_INDICATOR from the Column is *NULL*, otherwise *null.*

PROMPT the PROMPT field from the Column object.

USER_DEFINED_PROPERTY_0 is set to same value as the Column USER_DEFINED_PROPERTY_0 value.

USER_DEFINED_PROPERTY_1 is set to same value as the Column USER_DEFINED_PROPERTY_1 value.

USER_DEFINED_PROPERTY_2 is set to same value as the Column USER_DEFINED_PROPERTY_2 value.

USER_DEFINED_PROPERTY_3 is set to same value as the Column USER_DEFINED_PROPERTY_3 value.

USER_DEFINED_PROPERTY_4 is set to same value as the Column USER_DEFINED_PROPERTY_4 value.

USER_DEFINED_PROPERTY_5 is set to same value as the Column USER_DEFINED_PROPERTY_5 value.

USER_DEFINED_PROPERTY_6 is set to same value as the Column USER_DEFINED_PROPERTY_6 value.

USER_DEFINED_PROPERTY_7 is set to same value as the Column USER_DEFINED_PROPERTY_7 value.

USER_DEFINED_PROPERTY_9 is set to same value as the Column USER_DEFINED_PROPERTY_9 value.

5. *List of Files*

This wizard consists of the following set of files, all located in the *$DEVENV/D2K* directory.

d2k_dcu.sql	SQL*Plus script for updating Module Detail Column Usage information from Column information into the Designer/2000 repository. Used by wizard_mod.sh.
d2k_dcu_upd.sql	SQL*Plus script for updating Module Detail Column Usage information. Clearing DISPLAY_FORMAT and DISPLAU_LENGTH on DCU's referencing to DATE, TIMESTAMP or TIME columns.
d2k_dom2.sql	SQL*Plus script for updating Domain information to reflect the properties of the super domain.
d2k_dtu.sql	SQL*Plus script for updating Module Detail Table Usage information from Table information into the Designer/2000 repository. Used by wizard_mod.sh.
d2k_dtu_prf.sql	SQL*Plus script for updating Module Detail Table Usage information from secelted Preference information into the Designer/2000 repository. Used by wizard_mod.sh.
d2k_col.sql	SQL*Plus script for updating Column information from Attribute information into the Designer/2000 repository. Used by wizard.sh.
d2k_ent_map.sql	SQL*Plus script for creating Entity to Table definition Map information into the Designer/2000 repository. It assumes that the table_name is stored in the entity record as User-defined-property 1.
d2k_fk.sql	SQL*Plus script for creating and updating foreign key constraints and Key Component information from Relation information into the Designer/2000 repository.
d2k_inx.sql	SQL*Plus script for creating and updating Index and Index Entry Information from foreign key constraints and Key Component information into the Designer/2000 repository. Used by wizard.sh.
d2k_mod.sql	SQL*Plus script for updating Module information in the Designer/2000 repository. Used by wizard_mod.sh.
d2k_mod_usg.sql	SQL*Plus script for updating Summary Module Table Usage information in the Designer/2000 repository.

Dansk Data Elektronik A/S
Herlev Hovedgade 199
DK 2730 Herlev
Tel.: (+45) 42 84 50 11
Fax: (+45) 42 84 52 20