Accessing the Designer/2000 Repository Tables

Jeffrey M. Stander

Director & Principal Consultant

ANZUS Technology International.

Presen ted at Oracle Openworld Australia 1996

Accessing the Designer/2000 Repository Tables[1]

OBJECTIVES

- To present a basic understanding of the views and underlying tables which constitute the meta-model structure of the Repository Database Design Model.
- To present techniques that will enable a database designer to create custom views based on the repository views and tables which can be used to size database objects, analyse tablespace requirements, and create storage definitions for each object.

ABSTRACT

The underlying views and tables in which the Designer/2000 Repository stores it's information are confusing at first glance, but with a little understanding of the Repository Database Design Model it is not difficult to create custom views and SQL statements that can be used to:

- create DDL directly for generating tables, views, indexes and constraints which are defined in the Repository
- link to a spreadsheet to provide the basis for interactive tablespace, table and index size analysis
- easily create custom reports (e.g. table-column usage, index usage, etc.) in text format or linked to a spreadsheet or word processor without resorting to the Report Generator
- create tools for validating a database against the repository.

This paper gives an overview of the Repository Data Model, discusses how to create custom views on the Repository Data Model, and presents examples of the above applications.

[1] This paper is available on the website of the Tasmanian Oracle Users Group: http://lad.med.utas.edu.au/toug or from the author at willstand@acslink.aone.net.au.

INTRODUCTION

Oracle's Designer/2000 is a CASE tool that attempts to include enough functionality to serve as the basis for design or re-engineering of any database. It can be used, among other functions, to model business functions, design and generate forms and reports modules, and estimate module development time. This paper focuses principally on the use of Designer/2000 as a device for Entity-Relationship modelling and the construction of a physical database from that model.

Designer/2000's Repository Object Navigator (affectionately called "The RON") is the principal tool for defining these functional elements. Primary Access Component (PACs) such as tables or entities and Secondary Access Component (SACs) such as relationships or columns are defined with the RON using a hierarchical list (the Application Window) and a Property Sheet window for each component type. Diagramming Tools allow creation of ER diagrams and data diagrams which are extremely useful for analysis and programming. Elements may be created with the diagramming tools as well as in the RON (this is especially useful for creating view definitions).

Repository Reports are provided to print summary information or reconciliation's on the repository model, but I often found them too detailed, not detailed enough, or simply not useful, which Is why I undertook this exercise.

THE DATABASE DESIGN MODEL - VIEWS AND TABLES

The Designer/2000 Repository stores it's information in tables and views called the *Meta-model* (the model of the model). These are found in the repository owner's schema. Normally the user is expected to use only the views to access repository data.

For the Database Design Model we need to look at three tables:

SDD_ELEMENTS,
SDD_STRUCTURE_ELEMENTS,
and CDI_TEXT.

SDD_ELEMENTS defines the individual elements and the many-to-one relationships between them.

SDD_STRUCTURE_ELEMENTS is a many-to-many resolution table to relate elements, e.g. a single view has several base tables; a single table may belong to several views.

CDI_TEXT contains the text for WHERE clauses, free-format view SELECT statements, check constraint definitions, etc.

SDD_ELEMENTS is a massive table with cryptically-named columns and dozens of self-referencing foreign keys. It is usually easier to avoid this (and other) repository tables and make direct access the repository via the many repository views which are defined in the Designer/2000 system. Many of these views are defined as updateable views on a single table. By looking at the view text the base tables and column mapping can be ascertained. (But **NEVER DIRECTLY MODIFY THE REPOSITORY**. This should only be done using the Designer/2000 API). The on-line documentation also has column details and other information on the repository views in the section entitled *Application Repository Programmatic Interface.*

For example, CI_COLUMNS is a view on SDD_ELEMENTS which selects all the columns of all tables defined in the repository. A `SELECT` on this view can return column information with the `WHERE` clause restricting the output to a particular column, a particular table, and a particular Application System. Some of the foreign keys defined in this view are *table_reference* to CI_TABLE_DEFINITIONS, *sequence_reference* to CI_SEQUENCES, *source_attribute_reference* to CI_ATTRIBUTES.

The major relationships are found in the five data diagrams which are shipped with the Designer/2000 product. The one we are most interested in for this paper is the Database Design Model and a portion of this diagram is reproduced in Figure 1.

These underlying views and tables are daunting at first glance, but with a little understanding of the Repository Database Design Model it is not difficult to create your own custom views and SQL statements which may be used to

- directly create DDL for generating tables, views, indexes and constraints which are defined in the Repository
- link to a spreadsheet to provide interactive tablespace, table and index size analysis
- easily create custom reports (e.g. table-column usage, index usage, etc.) in text format or linked to a spreadsheet or word processor without resorting to the Report Generator.
- create tools for validating a database against the repository.

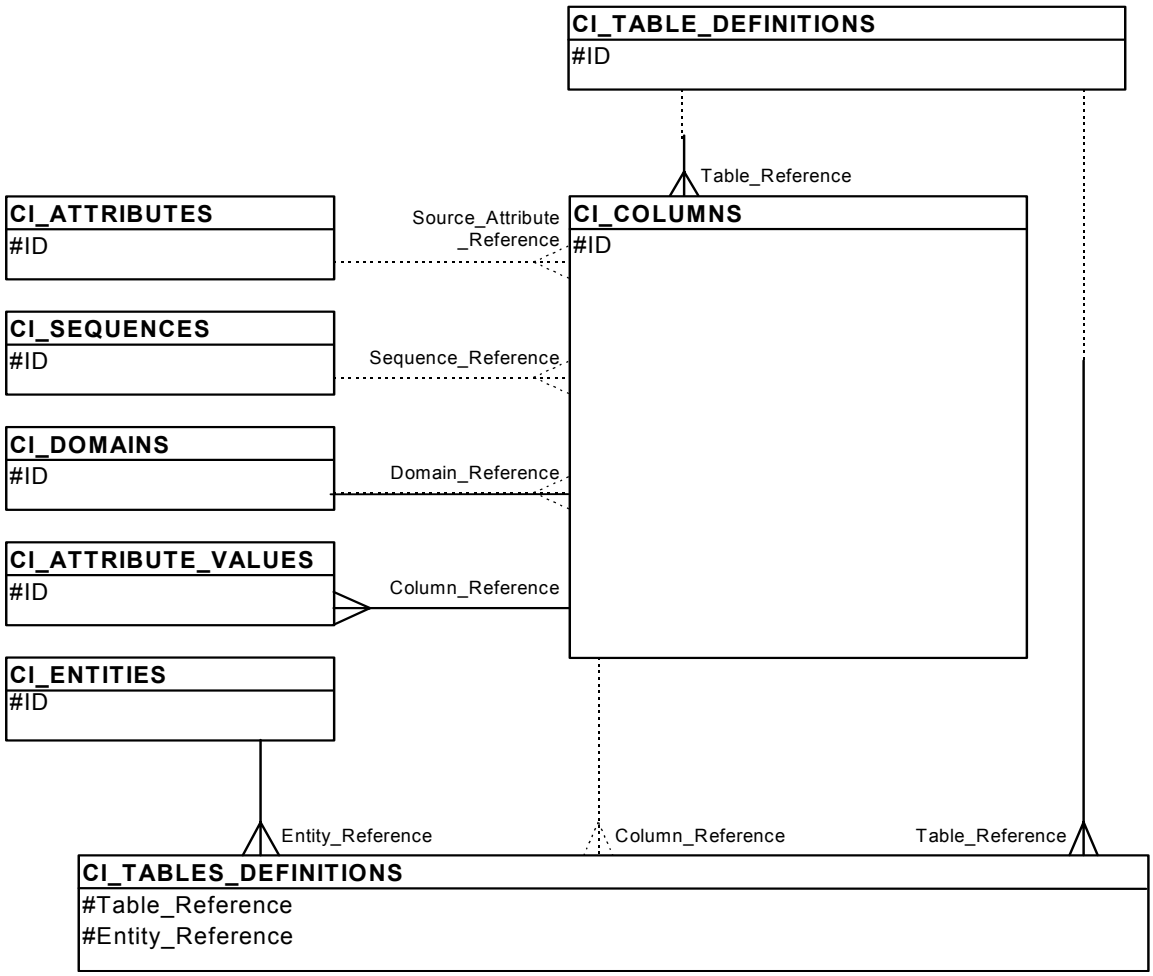

Figure 1. Partial Data Diagram of the Database Design Model

- create tools for validating data prior to enabling constraints.[2]

Figure 1 is a partial copy of the Database Design Model data diagram which is distributed by Oracle with the Designer/2000 product. Primary key fields for each view are given in the diagram and the principal foreign key relationships are drawn.(Note that this being a data diagram the crowfoot on the relationship line always represents the foreign key (child) end of the relationship and says nothing about whether it is a many-to-one or one-to-one relationship).
While this diagram is the basis for a good start, I have found it useful to extract the view text for the major views so I know how the view `SELECT` works and what the base column references are. It is sometimes easier and faster to directly access the base table instead of going through the view. This has it's risks, where it is unlikely that the view definitions will change during an upgrade, there are no guarantees that the base table will change (although I find it unlikely).

HOW TO USE THE VIEWS – AN EXAMPLE

When I first joined the project, some preliminary work had been done on creating a physical database from the ER model. Not being sure of what had been done already, one of the first things I had to do was to was to determine which attributes were not represented in the physical database model and which columns were not represented in the logical database model. Normally, if tables are created from the logical model using the Database Design Wizard (DDW) tool, there is an entity-table mapping and thus each column is an instance of a relationship (i.e. a foreign key) or an attribute. Sometimes entities or tables are subsequently modified by hand and the table update is neglected. How to reconcile the differences? The example given in Figure 2 is part of the answer — a query that finds columns

[2] 1996, *Cross-loading of Legacy Data Using the Designer/2000 Repository Data Model*. Jeffrey M. Stander ODTUG CASE Day 3 Nov 96 at Oracle OpenWorld, San Francisco, USA..

without any corresponding attributes or relationships. It is not as bad as it looks. The Application System is the database system under development; it's end product is a schema. (Note that Application Systems in the RON can share PACs between them, and ownership of PACs can be transferred). Application Systems have a name and a version number, which must be specified as shown in the example where ERD(2) is name and version number of the Application System. Thus CI_APPLICATION_SYSTEMS is correlated with the entities CI_ENTITIES view. The CI_TABLES_ENTITIES joins tables to entities and the WHERE clause with its subquery filters out any column which has a source attribute or is joined to a relationship.

```
-- Find columns without a source attribute or relationship reference

SELECT ENT.NAME ENTITY_NAME,
       TAB.NAME TABLE_NAME,
       COL.NAME COLUMN_NAME
FROM   CI_APPLICATION_SYSTEMS APP,
       CI_ENTITIES ENT,
       CI_TABLES_ENTITIES MAP,
       CI_RELATION_DEFINITIONS TAB,
       CI_COLUMNS COL
WHERE  APP.NAME = 'ERD'                          -- application system
AND    APP.VERSION = 2                           -- version number
AND    APP.ID = ENT.APPLICATION_SYSTEM_OWNED_BY
AND    ENT.ID = MAP.ENTITY_REFERENCEj
AND    COL.TABLE_REFERENCE = TAB.ID
AND    COL.SOURCE_ATTRIBUTE_REFERENCE IS NULL  -- not an attribute
AND    NOT EXISTS (SELECT NULL                   -- not a key component
                   FROM   CI_KEY_COMPONENTS KEY
                   WHERE  KEY.COLUMN_REFERENCE = COL.ID)
ORDER BY ENT.NAME, TAB.NAME, COL.NAME
/
```

Figure 2. Example of Using Repository Views to Look For Missing Links between the Logical and Physical Models.

THE DES2KUTL.PKG PACKAGE

As I began to create more SQL statements and views based on the repository views it became tiresome to always correlate CI_APPLICATION_SYSTEMS. I also wanted to write generic SQL which didn't hard code in the ERD(2) application system. So I began writing what became DES2KUTL.PKG (See Appendix 1), a package of inline functions which can be inserted into a SQL statement. For example, **des2kutl.app_id** without arguments returns the ID of the ERD(2) application system because it is hard-coded into the package. I just have to change the package definition and all scripts will refer to a new application system. In the above example,

```
WHERE  APP.NAME = 'ERD'
AND    APP.VERSION = 2
AND    APP.ID =
       ENT.APPLICATION_SYSTEM_OWNED_BY
```

is replaced with

```
WHERE ENT.APPLICATION_SYSTEM_OWNED_BY
      = DES2KUTL.APP_ID.
```

Inline functions are a very powerful tool for simplifying and/or enhancing SQL statements.

THE DES2KUTL.SYS VIEWS

As I worked with Designer/2000 and with the process of building the physical database I found I had repeating tasks that became difficult to perform. For instance, if I had to recreate a table on the development database it was necessary to drop the foreign key constraints that referenced that table as a parent table. Although this could be done using the USER_CONSTRAINTS table I wanted to also recreate the constraints afterwards. By writing two custom views, **fk_v** and **fk_cols_v,** I was able to do this. **fk_v** (Figure 3) selects the table name, constraint name, and foreign table name for all server-implemented foreign keys. **fk_cols_v** returns information on

```
-- view to list foreign keys with server-side
-- implementation and create status

prompt fk_v
create or replace force view fk_v
as
SELECT fk.el_id id
       ,substr(tab.el_name,1,30) table_name
       ,substr(fk.el_name,1,30) key_name
       ,fk.el_flag6 mandatory_flag
       ,substr(ft.el_name,1,30) foreign_table_name
FROM   sdd_elements tab
       ,sdd_elements fk
       ,sdd_elements ft
WHERE  tab.el_elem_owned_by = des2kutl.app_id
AND    tab.el_type_of='TAB'
AND    fk.el_type_of='OCO'
AND    fk.el_occur_type='FOREIGN'
AND    tab.el_id = fk.el_within_id
AND    fk.el_status = 'Y'
AND    fk.el_switches in ('BOTH','SERVER')
AND    fk.el_2nd_within_id = ft.el_id
GROUP BY
       fk.el_id,
       substr(tab.el_name,1,30),
       substr(fk.el_name,1,30),
       fk.el_flag6,
       substr(ft.el_name,1,30)
/
```

Figure 3. Custom View on the Repository Lists Foreign Keys

the columns which are components of the foreign key and the primary or unique key columns which they are referencing in the foreign table. (See Appendix)
These views, similar views for tables, primary keys, unique keys, and several other custom views I keep in the file DES2KUTL.SYS and this is reproduced in APPENDIX 2.

To generate DDL I devised the technique of creating a view which consists of a set of SELECT statements joined by UNION ALL statements. Each SELECT statement returns a piece of the DDL text and sequencing data to align the text. UNION ALL is used go avoid the implicit sort/merge which is performed by UNION, i.e. it runs faster. As an example, I will use the **cc_txt_v** view which generates DDL to create check constraints from the repository (Figure 4, below).

A describe on this view shows it consists of columns TABLE_NAME, KEY_NAME, TEXT, TYPE and SEQUENCE_NUMBER. The TEXT column is the DDL text we are interested in. The TYPE column identifies each of the UNION-ed SELECT statements and serves to order the text. The sequence number is always zero except for the actual text as selected from the repository table CDI_TEXT. For example, if we want DDL for all check constraints on the EMPLOYEE table we would

```
SELECT    text
FROM      CC_TXT_V
WHERE     TABLE_NAME =
          'EMPLOYEE'
ORDER BY  KEY_NAME,
          TYPE,
          SEQUENCE_NUMBER;
```

The output from this statement, spooled to a file, will create the desired check constraint(s)

TABLE SIZE ANALYSIS

Without going into great detail I will mention that I did not find table sizing for a physical database well supported in the Repository Reports. What I did was to use a number of views on the repository which were able to return the table name, tablespace_name, number of columns, initial and final row counts, initial, final and maximum column sizes (maximum and final differ because maximum assumes all VARCHAR2 fields are fully utilised) and percent free. Some of this information (row counts, percent free) was be entered by the designer during the design phase, although defaults are taken if not present

```
prompt cc_txt_v

create or replace force view cc_txt_v
as
SELECT substr(ft.name,1,30) table_name,
       substr(fk.name,1,30) key_name,
       'Prompt Creating Check Constraint '
         || fk.name || ' on ' || ft.name text,
       0 type,
       0 sequence_number
FROM   ci_check_constraints fk,
       ci_table_definitions ft
WHERE  ft.application_system_owned_by = des2kutl.app_id
AND    ft.id = fk.table_reference  /* Foreign Table */
AND    fk.create_status = 'Y'
AND    fk.implementation_level in ('BOTH','SERVER')
UNION ALL
SELECT substr(ft.name,1,30) table_name,
       substr(fk.name,1,30) key_name,
       'ALTER TABLE ' || ft.name || ' ADD (' text,
       1 type,
       0 sequence_number
FROM   ci_check_constraints fk,
       ci_table_definitions ft
WHERE  ft.application_system_owned_by = des2kutl.app_id
AND    ft.id = fk.table_reference  /* Foreign Table */
AND    fk.create_status = 'Y'
AND    fk.implementation_level in ('BOTH','SERVER')
UNION ALL
SELECT substr(ft.name,1,30) table_name,
       substr(fk.name,1,30) key_name,
       '      CONSTRAINT ' || fk.name text,
       2 type,
       0 sequence_number
FROM   ci_check_constraints fk,
       ci_table_definitions ft
WHERE  ft.application_system_owned_by = des2kutl.app_id
AND    ft.id = fk.table_reference  /* Foreign Table */
AND    fk.create_status = 'Y'
AND    fk.implementation_level in ('BOTH','SERVER')
UNION ALL
UNION ALL
SELECT substr(ft.name,1,30) table_name,
       substr(fk.name,1,30) key_name,
       '      CHECK (' text,
       3 type,
       0 sequence_number
FROM   ci_check_constraints fk,
       ci_table_definitions ft
WHERE  ft.application_system_owned_by = des2kutl.app_id
AND    ft.id = fk.table_reference  /* Foreign Table */
AND    fk.create_status = 'Y'
AND    fk.implementation_level in ('BOTH','SERVER')
UNION ALL
SELECT substr(ft.name,1,30) table_name,
       substr(fk.name,1,30) key_name,
       '      )' || chr(10) || ')' || chr(10)
       || '/' || chr(10) text,
       99 type_number,
       0 sequence_number
FROM   ci_check_constraints fk,
       ci_table_definitions ft
WHERE  ft.application_system_owned_by = des2kutl.app_id
AND    ft.id = fk.table_reference  /* Foreign Table */
AND    fk.create_status = 'Y'
AND    fk.implementation_level in ('BOTH','SERVER')
UNION ALL
SELECT substr(ft.name,1,30) table_name,
       substr(fk.name,1,30) key_name,
       '      ' || txt.txt_text text,
       4 type_number,
       txt.txt_seq sequence_number
FROM   ci_check_constraints fk,
       ci_table_definitions ft,
       cdi_text txt
WHERE  ft.application_system_owned_by = des2kutl.app_id
AND    ft.id = fk.table_reference  /* Foreign Table */
AND    fk.create_status = 'Y'
AND    fk.implementation_level in ('BOTH','SERVER')
AND    txt.txt_ref = fk.id
/
```

Figure 4. View generates check constraint DDL.

(e.g. 10 for percent free).

The custom views calculate column overheads and sizing. The view results are loaded into an Excel spreadsheet either directly (GLUE or ODBC) or as a text file, and the spreadsheet then computes rows/block, number of blocks, and initial, final and maximum megabytes required. The first page of a size analysis is reproduced in Figure 5. Once in the spreadsheet form it is possible to see the impact of varying parameters such as block size (global), or percent free(individual tables). E.g. by setting percent free to 1 for tables which grew but were not updated we would save 400MB of space over six months.

Using standard Excel functions (SUM_IF, COUNT_IF) it was also possible to generate a summary table of usage by tablespace.

CONCLUSION

Space and time does not permit me to describe everything I have done with the views listed in the appendices. As well, there is the Repository API which is available for those who wish to programmatically alter the Repository, to load storage parameters for instance, and there is the option of User Extensibility which allows extending the functionality of the Repository through creating custom properties which can be used to control user-defined columns in the Repository tables and views.

The intention is to describe how one can peer behind the GUI screen and make the data in the repository work for you in the way you wish.

SOURCE AVAILABILITY

I will be happy to share the views, packages, and scripts mentioned in this article. Please contact the author by email. Also note that this paper and the sizing spreadsheet is available on the website of the Tasmanian Oracle Users Group located at http://lad.med.utas.edu.au/toug.

TABLE SIZING

Note: tablespace names were assigned in the RON as substitution variables and mapped to tablespaces at build time

Designer/2000 Table Sizing

Constants

ORABlockSize	4096
InitTrans	1
Transaction Entry	23
PCTFree	10
Table Entry	4

Derived Values

Block Header	136
DataSpace	3933

Table Volume Summary

Tables	143
Zero Initial Volume	34

This greyed area is populated by a SELECT from a view on the repository (see file **tabviews.sql**)

Initial and 2-year row counts are the values entered against the table for start and final row volumes. 6-Month value is an interpolation

Summary by Tablespace (M	Tables	Initial	6-Month	2-Year	Max
&ASSETHOLDING	1	29	33	44	105
&ASH_RELATED	10	15	19	31	48
&GENERAL	63	555	1166	2999	6914
&PARTY	4	239	342	647	2179
&PARTY_RELATED	20	78	128	276	427
&PRODUCT	1	165	174	198	533
&PRODUCT_RELATED	23	462	542	783	10711
&STATEMENT	15	62	68	88	142
&STATIC	6	2	2	2	49
Total	**143**	**1607**	**2472**	**5067**	**21108**

TABLES	COLS	ROWS			COLUMN SIZE			PCT	ROWS / BLOCK			BLOCKS				MEGABYTES			
		Initial	2-Year	6-Month	Initial	Final	Max	Free	Initial	Final	Max	Initial	6-Month	2-Year	Max	Initial	6-Month	2-Year	Max
DIARY_EVENT	14	3500000	4200000	3675000	120	120	210	10	29	29	17	118668	124601	142401	249202	463.55	486.72	556.25	973.44
PRODUCT_ACTIVITY	16	3500000	4200000	3675000	106	106	2151	10	33	33	2	104823	110064	125788	2552539	409.46	429.94	491.36	9970.85
DAILY_ACCRUAL_HISTORY	16	0	14000000	3500000	100	100	235	10	35	35	15	3	98890	395558	929562	0.01	386.29	1545.15	3631.10
PRODUCT	136	250000	300000	262500	599	599	1610	10	6	6	2	42311	44426	50773	136468	165.28	173.54	198.33	533.08
PARTY	63	360000	1000000	520000	245	247	1110	10	14	14	3	24920	36290	69788	313621	97.34	141.76	272.61	1225.08
PARTY_ADDRESS	25	800000	2000000	1100000	115	115	324	10	31	31	11	25994	35742	64985	183087	101.54	139.62	253.85	715.18
MAJOR_ISSUE_NOTE	11	10000	500000	132500	578	578	2109	10	6	6	2	1633	21638	81655	297940	6.38	84.53	318.96	1163.83
PARTY_NAME	11	350000	1050000	525000	100	100	199	10	35	35	18	9889	14833	29667	59037	38.63	57.94	115.89	230.61
COMMUNICATION	25	0	1000000	250000	186	186	546	10	19	19	6	3	13138	52553	154268	0.01	51.32	205.28	602.61
PARTY_PRODUCT_ROLE	13	500000	600000	525000	77	77	128	10	46	46	28	10878	11422	13053	21699	42.49	44.62	50.99	84.76
PARTY_COMMUNICATION	8	0	2000000	500000	62	62	96	10	57	57	37	3	8759	35035	54248	0.01	34.21	136.86	211.91
RECEIPT_ITEM	16	0	1400000	350000	88	88	161	10	40	40	22	3	8702	34809	63685	0.01	33.99	135.97	248.77
ASSETHOLDING	86	80000	120000	90000	333	333	793	10	11	11	4	7527	8468	11290	26887	29.40	33.08	44.10	105.03
PARTY_PARTY_ROLE	7	360000	1000000	520000	57	57	82	10	62	62	43	5798	8375	16105	23168	22.65	32.71	62.91	90.50
PRODUCT_STATEMENT_RECIPIENT	9	350000	500000	387500	71	71	112	10	50	50	32	7021	7773	10030	15822	27.43	30.36	39.18	61.81
PRODUCT_STATEMENT	14	300000	400000	325000	77	77	126	10	46	46	28	6527	7071	8702	14240	25.49	27.62	33.99	55.63
PRODUCT_REPORT_CAT	8	250000	600000	337500	61	61	86	10	58	58	41	4309	5817	10341	14579	16.83	22.72	40.39	56.95
PROVISIONAL_ASSETHOLDING	19	80000	100000	85000	220	220	673	10	16	16	5	4973	5284	6216	19015	19.42	20.64	24.28	74.28
MAJOR_ISSUE	12	10000	250000	70000	231	231	373	10	15	15	9	653	4569	16317	26347	2.55	17.85	63.74	102.92
PRODUCT_SOURCE_OF_REFERRAL	7	250000	300000	262500	57	57	82	10	62	62	43	4026	4228	4831	6951	15.73	16.51	18.87	27.15
CLIENT_NON_CMSN_FEE_ACCRUAL	23	0	500000	125000	117	117	318	10	30	30	11	3	4132	16529	44924	0.01	16.14	64.57	175.48
VENDOR_ACCOUNT_REFERENCE	8	100000	500000	200000	65	65	118	10	54	54	30	1837	3673	9183	16670	7.17	14.35	35.87	65.12
TAX_STATUS_HISTORY	11	120000	240000	150000	80	80	109	10	44	44	32	2712	3391	5425	7391	10.60	13.24	21.19	28.87
CLIENT_CMSN_FEE_ACCRUAL	12	0	500000	125000	88	88	196	10	40	40	18	3	3108	12432	27689	0.01	12.14	48.56	108.16
UPLIFT_INTEREST_HISTORY	11	0	500000	125000	87	87	181	10	41	41	20	3	3073	12291	25570	0.01	12.00	48.01	99.88
BENEFICIARY_ENTITLEMENT	19	105000	125000	110000	93	93	228	10	38	38	16	2759	2890	3285	8052	10.78	11.29	12.83	31.45
PAYMENT_SEARCH	10	20000	400000	115000	85	85	157	10	42	42	23	480	2762	9606	17744	1.88	10.79	37.53	69.31
PARTY_ASSET_ROLE	8	160000	180000	165000	56	56	91	10	63	63	39	2532	2611	2848	4628	9.89	10.20	11.13	18.08
PARTY_TFN	10	120000	145000	126250	73	73	106	10	48	48	33	2475	2604	2991	4343	9.67	10.17	11.68	16.96
INSURANCE	20	50000	60000	52500	147	147	315	10	24	24	11	2077	2181	2492	5340	8.11	8.52	9.73	20.86
LIABILITY	23	50000	60000	52500	140	140	259	10	25	25	14	1978	2077	2373	4391	7.73	8.11	9.27	17.15
BENEFICIARY_ASSET_RECIPIENT	7	105000	125000	110000	59	59	87	10	60	60	41	1750	1834	2084	3073	6.84	7.16	8.14	12.00
WILL_TRANSIT_CONTROL	11	70000	100000	77500	83	83	115	10	43	43	31	1642	1817	2345	3249	6.41	7.10	9.16	12.69
BANK_STATEMENT	11	20000	180000	60000	74	74	106	10	48	48	33	418	1254	3763	5391	1.63	4.90	14.70	21.06
PAYMENT_ITEM	11	60000	63000	60750	69	69	118	10	51	51	30	1170	1184	1228	2100	4.57	4.63	4.80	8.20

Initial column size depends on data type and start volume percentage as entered in the tables's property sheet in the RON

Max column size assumes each CHAR field is filled.

Final size depends on data type and end volume percentage as entered in the tables's property sheet in the RON.

Figure 5. Page from table sizing spreadsheet.