The following paper was originally presented at the
Ninth System Administration Conference (LISA '95)
Monterey, California, September 18-22, 1995

SQL_2_HTML: Automatic Generation of HTML Database Schemas

Jon Finke
Rensselaer Polytechnic Institute

SQL_2_HTML: Automatic Generation of HTML Database Schemas

Jon Finke – Rensselaer Polytechnic Institute

ABSTRACT

The ongoing development of our relational database based system administration package, Simon, requires frequent reference to documentation that describes the existing database tables. To this end we have written a program that uses descriptive information stored in the database itself, to generate a WWW tree that documents each table in HTML, as well as an index page to tie the whole package together. This has made looking up table definitions simply a click or two away and has proven to be very useful. These HTML pages are now also being included in some of our program documentation of the Simon system.

Introduction

We manage many of our Unix system administration tasks via a system we have developed called Simon. The Simon system is based on a relational database, in our case, Oracle. At the start of the project, the importance of documentation of the database design was recognized so we made use of an Oracle feature that allows you to store comments on both tables and columns directly in the database. We first wrote a program that would generate a simple text file with the description and attributes of each table. As the number of tables grew, this program was modified to generate *man*(1) pages.

Along with the man pages, versions of this program were produced to generate both TeX and nroff format descriptions of the database objects for inclusion in papers and other documentation. This has served to strengthen our practice of requiring that all tables and columns be fully documented in the database before they are installed in the production system. This also prevents us from loosing the documentation, as long as the table lives, so does the documentation.

Unfortunately, with the number of tables that make up the Simon system, the man(1) based solution was not scaling well; often the key piece of information needed was the table name, and the man(1) command demanded an exact match. With the growing availability of WWW browsers, a new solution appeared. The program that generated the man pages went through yet another evolution to become **SQL_2_HTML** a program that generates HTML descriptions of each database table, and the key, an index of all the tables with hot-links to each.

HTML File Generation

For each table SQL_2_HTML is processing, we create an HTML format file. All the files are written to the current directory with the file name being the table name (with the first letter in each word of the table name capitalized), and a suffix of ".html". By following a consistent file name format, it is simple to generate references to each file from other html pages. Since all the files are in the same directory, we can use relative links to switch between tables without any worry about the actual file system path. Thus, we end up with some files like the following:

```
Disk_Acct.html Logins.html People.html
```

to hold documentation for the Disk_Acct, Logins and People tables.

Table Information

Once the file is created, we first write an HTML title and header lines, followed by the description of the table itself, as stored in the table comments part of the database. We can optionally include the table creation date and last time the table definition was changed. Another option will include the number of rows in the table, and the amount of space taken by the table. This is of course the size at the time the html page is generated (and should include the date of generation.)

It is possible to define an index on one or more columns in a table. This can provide some performance improvements by allowing the database to just traverse the index to find the desired row, rather than having to read the entire table. However, depending on the nature of the data, an index may not always help performance, and may even hurt performance in some cases. Thus, when investigating the performance of a query on a table, it is important to know what indexes exist for the table. Therefore, we consider all of the indexes on table an important part of the documentation for the table and include a list of those indexes in the HTML page.

Each index is listed by name, and with the column or columns that are indexed. As an added convenience, each column name is also a link to the specific column description, which follows later in

the HTML document. This is helpful as in some of the larger tables, the list of columns can be quite long.
This can be seen in Figure 1.[1]

One of the keys to generating source code for any text processor, including the web, is to provide proper processing for special characters. Since the people who entered the comments on tables and columns were not concerned about special characters, any comments (or other info) extracted from the database needs to be "cleaned" in order to trap the special characters. To do this, we wrote a version of `fprintf` that expands the arguments into a buffer, scans the buffer for special characters ("<", ">'' and "&'') and quotes them properly, and then calls `fprintf` with the "cleaned" string. Of course when we need to actually generate HTML directives, we have to call `fprintf` directly. We used a similar technique in the programs that generated LaTeX and troff documents.

Column Information

Now that we have extracted most of the general information for the table, we generate an entry for each column in the table. Along with the column name, we obtain the datatype definition for that column[2] and use these as the title (<dt>) in an HTML descriptive list (<dl>..</dl>). To assist readers in distinguishing the name from the type, we also put the column name in bold face type. We also define an anchor point at the column name, using a lower case version of the column name as a link name. This allows links to be made to the column information from within this page, or from other documents. Since we use a consistent name pattern a reference to the Username column in the Logins table would be written:

```
href="Logins.html#username"
```

The fundamental concept of a relational database is its ability to *join* two tables together using a column from each of the tables as a key. For instance, in the Logins table, there is a column called OWNER which indicates the owner of that particular Unix Login. Rather than storing the owner's name, addresses, etc, the OWNER column contains a numeric value (or key) which corresponds to the People.Id[3] column. Since many tables refer to the People.Id column and we have taken measure to ensure that all valid People.Id values can be found in the People table; the People.Id column is considered to be the **primary key**[4] of the People table. Since the Logins.Owner column refers to the People.Id column, the Logins.Owner column is considered a **foreign key** in the Logins table.

The words `Primary Key` in the heading after the data type indicate that that particular type is a primary key for that table. Alternately, if the column is considered a foreign key, this is indicated with the words `Foreign Key` followed by the name of the primary key. Since we follow a consistent pattern in file naming and the column anchor

[1]The examples shown here were displayed using Netscape Version 1.1.
[2]Number, Char(nn), Date, etc

[3]We often refer to database objects as a format like {Table}.{column} or {owner}.{table}.{column}, thus People.Id is the ID column in the People table.
[4]This is not intended to be rigorous definition of "Primary Key"

Oracle TABLE Logins

All past and present accounts, as well as reserved username and uids.

When Created: Sat Sep 10 23:06:02 1994
Last Modified: Tue May 16 17:50:23 1995
Number of Rows: 16117
Kilobytes used: 2684
Index: I_Logins$Owner Logins (owner)
Index: I_Logins$Unixuid Logins (unixuid)

username char(8)
    8-character identifier, also called login name, userid, login, netname, or account; key for this table, but not necessarily unique, given different source values
unixuid number - Primary Key
    Referenced by: Group_Members.Unixuid, People.Employee_Uid, People.Student_Uid,
    UNIX UID; should usually be between 1000 and 32767; NULL if not UNIX
public_personal_info char(240)
    Public personal information as it should appear in system list (aka gecos)
owner number - Foreign Key: People.Id
    People.id of the person currently responsible for use; usually the only person who knows the password

Figure 1: Sample Table

points, we are able to make the primary key name a hot link to the actual primary key definition.

Now that we finally have the title of the column entry set, we can work on the actual descriptive entry (<dd>). If this column is a primary key, we first list all the foreign keys that refer to it. Each of these entries also act as a hot link back to their own table/column definitions. After that, we include any comments that have been saved for that column.

Relational Links

The table and column comments, and documentation generation programs have been with the Simon project from the start, however no effort was made to formally record the relationship between primary keys and foreign keys. In fact, the *SQL Language Reference Manual (version 6.0)* states:

> Currently ORACLE Version 6.0 supports the syntax of constraints, and stores all constraint definitions in the Data Dictionary. This version does not actually enforce constraint definitions (NOT NULL is currently supported.)

Until the advent of commonly available hyper-text browsers such as Netscape and Mosaic, we did not bother to record the relationships (except as text in the actual comments.)

With the start of the **SQL_2_HTML** program, these constraint definitions suddenly became relevant. Rather than trying to parse column comments for other column names[5] we can now document the **relationships** between columns and tables directly in the database. This has allowed us to not only put in links from a column to the primary key in another table, but for each primary key, provide a list of tables (with links of course) that reference it.

Fortunately, we can go back after the fact and start recording these constraints.
We can define a primary key (say for the people table) as follows:

```
Alter table People
   add (primary key ( Id ) )
```

and then set Logins.Owner as a foreign key that points to it with:

```
Alter table Logins
   add (foreign key  ( Owner )
          references People.Id)
```

Views and Sequences

Along with the tables and indexes already described, we make use of a number of *Sequences* and *Views.* As with tables, we want to document them.

[5]Yes, I was considering that option for a while. Fortunately we came up with this instead.

A sequence is like a table with two columns defined, "currval" and "nextval''. When you select from a sequence say "Uidcount.Nextval", you will get a number that is one higher than the previous time you made that selection. This is very handy when you need a source of numbers, such as for assigning Unix Uids. In fact, sequences have a number of attributes including the step size, the direction, the maximum value, the minimum value, and if the sequence "wraps" when it hits the maximum. This can be very handy when you want to stop some processing when it hits a limit, such as running out of Unix Uids to assign.

Generation of a sequence HTML page is straightforward. We use the same file naming convention that we use for tables. Unfortunately, oracle[6] does not currently support comments for sequences. This limitation could be addressed with the creation of a table and a few views that would virtually duplicate the table/column comment functionality with only some minor syntax differences. All of the interesting information on the sequence is extracted from a table and put into the HTML file. A sequence page is shown in Figure 2.

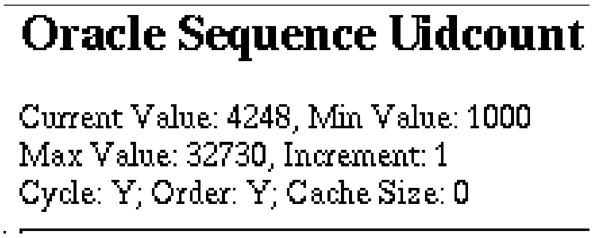

Figure 2: Sequence Page

Another very useful database object is called a view. This is a logical table based on one or more real tables. Views allow you to grant other people access to part of a table, based on any number of constraints you can define. Since views are essentially virtual tables, generation of an HTML page for a view starts out much the same way as we do for a table. The view (table) comments, the creation and last modified dates are all included, as well as the number of rows. Since views don't actually store any data (all the data lives in the real tables), there isn't any value to use for the space used.

Internally, a view is a database query of some type. While it can make use of indexes that may exist on the actual tables, you can not create an index on view, so there are no indexes to be included on the view pages. Another difference, is that you can not create constraints on the columns in a view, so the column information is limited to the column name, the data type and the column description.

[6]Oracle version 6.0.37

One important thing that is missing from the view pages, is the actual definition of the view. Due to a limitation in the interface used, this information was not available for this version of the program. Once this restriction is lifted, not only will we be able to include the definition of the view on the HTML page, but we will be able to reference the underlying tables for additional comments and constraints.

Table Index

Once we have generated files containing HTML descriptions of each table, sequence and view, we then build an index page to help us get around. The index is split into three sections, one for tables, one for views and one for sequences. Each table name (or view name, or sequence name) is a link the the corresponding HTML page. The index is currently generated as a descriptive list, with the table/view name as the title (<dt>) information, and the table description as the list data (<dd>).

Even though Oracle limits the table comments to 250 characters, with the number of tables growing, the index of tables is getting to be pretty long and is becoming difficult to scroll through. We have *jump bars*[7] to get you to the start of any of the sections, but the table and view sections are simply getting to long. A short list of all tables, views and sequences without the descriptions may be useful for quick access. A look at part of the index page, as it goes from tables to views (including a jump bar) is in Figure 3.

[7] Jump bars are those horizontal lists of hotlinks in an HTML document that help you navigate within large documents. They are repeated throughout the document so you don't have to scroll too far before you can click and jump.

Future Directions

There are a number of changes and extensions that I would like to do with this program. Some are basically just finishing the existing work, and others will take some more design to find the best approach.

Improve View Support

As mentioned in the section on views, I want to extract the definition of the view and include it in the page. This was not done in the initial version due to a limitation in the API[8] we use to access the database. Once we are able to extract the view definition, we want to parse the definition to identify the base tables and columns and make those as hot links to the appropriate places.

Be able to access the underlying column definitions will also allow us to extract the column descriptions to be included in the view documentation. We often did not take the time to document the column descriptions in a view since their description was unchanged from the base table.

[8] We use a locally developed API called RSQL. It is a subroutine interface we developed to connect to the vendor application interface (OCI). In many ways it is much simpler to use than OCI, and allows us some additional networking options. On the other hand, it does not currently support the LONG datatype, which is how view definitions are stored. Since SQL_2_HTML is the first application that needs this, we may finally add the support.

Userid_Changes

A list of pending and completed requests to change RCS userids from one value to another. We assume that any pending changes have met all policy and ownership requirements.

User_Directory_Info

A table of user directory information that people may want to update for inclusion in the directory. The one row per user is enforced via an unique index

Volumes

The list of volumes that we know about and bill for. Entries are added when we detect new volumes by comparing this to afs_volumes, updated when a quota changes, and during billing.

Wtmp_Status_Log

A place to record the last time a host checked on the wtmp_status version, and the last time that version was updated.

Jump to: Tables Views Sequences

Oracle Views for Simon

Etc_Passwd

A view of the logins table, roughly equivelent to what could be found in the /etc/passwd file. It does have the addition of the SOURCE field and the OWNER field. Passwd hash is not available. Only active userids are returned.

My_Mail_Group_Name_Memberships

A view of all the mail group names that this person is in some way a member of. While alias names are normally non published, being a member of a mail group entitles you to know the name. Most columns come directly from the Mail_Group_Names table

---

Figure 3: Sample Index

Weak Links

In a perfect relational database, you never have more than one copy of any given data element. In practice however, I find that data is often copied between tables, especially when interfacing to some external agency. For instance, when we match the Simon student list with the Registrar's student list, we use the "student number"[9]. This is later copied to the Simon.People table, and appears in a few other places as well. No one table really "owns" the student number, yet many different table reference it. Since we can't make it a primary key, we could define a "weak link" table, to help hold these relationships together in the HTML page.

We also have relations between tables. The Simon.Students and Simon.Employees tables both feed into the Simon.People table. Since this data propagation is often of interest, being able to generate links that follow these paths would also be useful. Since the table HTML pages can be regenerated at any time, these relations would also have to be stored in the database. If this was done effectively, this could also form the basis for a data flow diagram.

Multiple Schema Support

While most of the Simon tables simply reference other Simon tables, we do have other projects that reference Simon tables, and some Simon table make references to other projects. However, due to operational requirements, we can't have all the tables from all the database users in the same directory. Not only do you have the problem of file name conflicts (although that could be solved by prepending the Owner to the table name), you have licensing restrictions to consider. While it may be fine for RPI to publicly permit the Simon schema, it would be questionable at the very least to publicly permit the schema for the Financial accounting system that we run.

One solution to this problem, would be to create a html_doc_home table that records the full path[10] or URL prefix for each Schema. In this way, when SQL_2_HTML is generating a reference to a table/column in a different schema, it can look in this table to get the appropriate prefix to make the hot link.

Database Access Status

One set of information that would be very useful when administering the Simon system, would be the table access information. This would have a list for each table and view, of who has what kinds of access. It would also have a page for each Simon user of what specific access they have.

While easy to generate, I have not yet decided if this should be included on the page with each table, or if it should be broken out into it's own tree. We need to consider the privacy and security issues involved here. Providing a potential hacker with a list of who has access to sensitive information would make it easier for them to target an attack on an individual.[11] Given the operational rather then development nature of this information, this might be better kept elsewhere (with links to the main tree of course.)

External References

Programs outside of the database also reference tables and columns. It would be nice to be looking at a particular table or column, and find out what programs and subroutines access it. Ideally you would want to be able to link to both documentation and source code[12].

On the assumption that you are going to document your programs and subroutine libraries, it is a simple matter to include links to the tables. However, I find I often want to go the other way, given a table, find out who references it. One approach to this, is create a Reference Registry table, where you can add references to the document when you are writing it, and when the table document is regenerated, these references are automatically included. Ideas on how to automate the registration would be welcome.

Including references to the source code might be a bit trickier. Currently, just about all of the Simon code[13] uses the RSQL routines to make oracle calls. These look just like `printf` statements, so it would be possible, sort of, to parse these to identify tables and columns references. Unfortunately, some of the statements use variable substitution, so you have execute the code to actually see what tables and columns are used.

```
sql("Select X,Y,Z");
sql("  from Simon.Table");
```

works, while

```
char *third_col, *tab_name;
sql("Select X,Y,%s", third_col);
sql("  from %s", tab_name);
```

requires actual execution to identify the tables and columns.

[9] A 9 digit number frequently mistaken for a social security number.

[10] At RPI, many of our systems share a common AFS file system, so a file system path is adequate for many of the references. Also, some pages can not be released to the public, so we rely on the file system access control.

[11] Security by obscurity is not a good idea, but this might keep some less obvious targets hidden.

[12] After all, isn't the source code the ultimate program documentation?

[13] At last count, around 50,000 lines of code in about 200 modules.

The idea of "scanning" the code for references does still have some appeal. A way of building a general program cross reference to trace subroutine calls would be handy. Some of this may already be handled in other software, or in packages like "ctags" for emacs. For now I may have to be content with simply documenting what I write in HTML, and making the links to the tables.

Another alternative would be to embed references in comments such as:

```
/* SQL_TABLE(Simon.Logins,Insert) */
/* SQL_TABLE(Simon.People,Delete) */
/* SQL_COLUMN(Logins.Owner,Update) */
/* SQL_COLUMN(People.Id,Select) */
```

This actually includes not only the table/column information, but some indication of what action may take place (Select, Insert, Update, Delete). This could be very useful in determining which program update or reference tables. Unfortunately, it requires adding additional comments to the programs.

Availability

As with all other parts of the Simon project, this program is available to anyone who wants it. The actual Simon HTML tables are available from a link on my home page. While we do not have a formal release, all Simon and supporting code are freely available via AFS or anonymous FTP. These are the actual working directories.

For anon FTP, connect to ftp.rpi.edu, and root around in the pub/its_release directory. In the "simon" directory, there is a README file that explains what is available and where to find it. For AFS users, take a look in /afs/rpi.edu/campus /rpi/simon.

The SQL_2_HTML program currently requires the RSQL routines. These are available for Oracle Version 6, Oracle version 7, and another site has ported them to Sybase.[14]

References

[ORACLE90] "SQL Language Reference Manual; Version 6.0", Oracle Corp, Revised February 1990..

[FINKE92] "Oracle Tools", Jon Finke, Community Workshop 92, hosted by Rensselaer Polytechnic Institute, June 13-19, 1992, Troy, NY.

Author Information

Jon Finke graduated from Rensselaer in 1983, where he had provided microcomputer support and communications programming, with a BS-ECSE. He continued as a full time staff member in the computer center. From PC communications, he moved into mainframe communications and networking, and then on to Unix support, including a stint in the Nysernet Network Information Center. A charter member of the Workstation Support Group he took over printing development and support and later inherited the Simon project, which has been his primary focus for the past 4 years. Reach him via USMail at RPI; VCC 315; 110 8th St; Troy, NY 12180-3590. Reach him electronically at finkej@rpi.edu. Find out more via http: //www.rpi.edu/~finkej.

[14] At that time, Sybase did not have the table and column comment facility, although this would be trivial to implement with a few tables and views.