

Autojoin: A Simple Rule Based Query Service for Complex Databases

Niall I. Gaffney, Lisa Gardner, Molly Brandt

Space Telescope Science Institute, 3700 San Martin Drive, Baltimore, MD 21218

Abstract. Most databases used today are no longer flat. While the power of using these more complex data stores is well known, construction of queries can be quite a complex task. Currently this often requires detailed knowledge of the database structure and schema. As we move towards a VO paradigm, users cannot be expected to know the structure of databases, but will need to query them. Databases will need to provide query engines to complete queries automatically given only what the user wants to have returned and any qualifications they place on the query.

For years StarView, a database query and data retrieval tool for the Space Telescope Science Institute, relied on a complex third party LISP-based program (QUICK) to construct valid SQL queries for the one database it could query. This limited our ability to support StarView as we could not easily add new rules to the system without completely rebuilding the query engine. Furthermore, QUICK did not have the ability to create SQL that would join tables in different databases (but hosted on the same server). Finally, the cost of upgrading to a new version of QUICK was prohibitively high.

Our solution was to develop a rather simple database table driven Perl CGI program which is able to take as its input a skeleton SQL program. This may come from a program or other web page. In the query only the SELECT and user qualified WHERE clause are specified; no FROM or WHERE clause join information is included. The service then returns a fully qualified and syntactically correct query for the host database SQL program that can be used to get the information the user needs. Thus, an additional layer of abstraction for dealing with databases is created, freeing the user from having to know how tables are related in the database.

In this paper we discuss the design and algorithm used to make Autojoin work as well as discuss how, when combined with a robust and searchable description of all the fields that can be publicly queried in the database, it allows users to tailor their questions to the database with ease and a high rate of success.

A subset of STScI Proposal and Data Archive Database structure

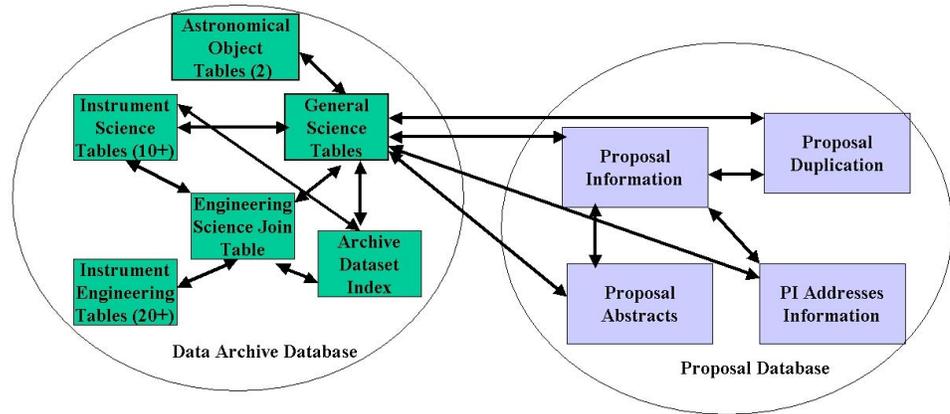


Figure 1. A representation of the database schema at STScI.

1. The STScI Database: The Task at Hand

At STScI, information about observed and planned observations is stored in several databases. These databases are rather large, and have many different rules for joining tables in one database, much less between separate databases. Different subsystems often map into the database schema differently, leading to complex intersystem data structures. Figure 1 gives a small example of the complexity of this system for a selection of tables from two of the STScI databases.

As illustrated in Figure 1, there are many different routes to join related information. To create a join from the Proposal Abstract to information in an Instrument Engineering table requires several interim join tables. Users cannot be expected to know this.

Laying out the database structure in this way reveals clear patterns in how the database works. When one looks at the keys that are used to join these tables, further patterns develop. It was clear from this exercise that a simple rule based system for creating arbitrary queries to the database could be constructed. We started by cataloging all the tables to be exposed to users. We then categorized these tables to group tables that join to other tables using the same keys. Finally we constructed rules for joining between different categories of tables. For some tables, interim join tables were required, and noted in the join rules.

2. Autojoin: The Tables Used to Find Joins

With the categorization done, we created three tables to capture this information. The Table Categories table contains the list of all publically available tables and the databases that house them. For some databases at STScI a prefix

for each table is designated to allow for a namespace differentiation of shared fields among different tables. For example two tables might have an RA field with prefixes `one_` and `two_`. The field in table one would be called `one_ra`, while the other is `two_ra`. This prevents database field namespace conflicts in these complex databases while preserving shared field names for common fields. The last field sorts these tables into categories. Tables in the same category join with other tables the same way.

The Category Joins table lists all the possible joins between two table categories listed in the Table Categories table. Each join is assigned a join number. The rules for how to join these two tables are listed in the Join Methods table, where the join number from the Category Joins table indicated which rule to use for joining those tables. Examples of these tables follow.

Table 1. Table Categories—public tables and their join categorization.

database_name	table_name	table_prefix	category
catalog	acs_a_data	aca_	hst-cal-exp
catalog	archive_data_set_all	ads_	hst-ads
catalog	science	sci_	hst-sci
proposal_db	abstract	NULL	hst-prop

Table 2. Table Joins—possible joins between the different table categories.

category_1	category_2	join_number
hst-cal-exp	hst_sci	18
hst-ads	hst_sci	1
hst-prop	hst_sci	3

Table 3. Join Methods—the stubbed out SQL for the where clause for each possible join number or the name of the join table to be used in the interim (as indicated by the leading #).

Join Number	Join Clause
1	\$table1.{prefix}dataid = \$table2.{prefix}dataid AND \$table1.ads_program_id not NULL
3	\$table1.{prefix}proposalid = \$table2.{prefix}propid
18	#catelog.sci_inst_db_join

3. Autojoin: The Algorithm

The autojoin algorithm takes these three tables and finds a path that uniquely joins each table. It is:

- Gather the list of tables to process from the initially supplied SELECT and WHERE statement
- Reorder input list sorted by their categories in Table Categories. This is done so that creating the join takes a minimum number of steps (similar table types will not join to a different type of table using an intermediate table except where needed).
- Create a new WHERE clause list and FROM list to store all the tables in the query and the new WHERE clause items as we discover them
- Loop through each pair of tables (i.e., 1st and 2nd, 2nd and 3rd)
- Swap in order if order in Category Joins is reversed
- Exit loop with an error if there is no entry in Category Joins for the two tables.
- Interpolate the join using Join Methods for that join number.
- Exit loop if both tables are the same or if new WHERE clause item appears in the new WHERE clause array.
- Add the new WHERE clause elements to the WHERE array and add the two tables to the from array.
- If dealing with an intermediate table (as identified by leading # in the Join Method entry):
 - If this intermediate table had already been used to join other tables in this query, join these tables to the other tables that used the intermediate table.
 - If this intermediate table had not already been used, join the intermediate table to the two current tables and add the intermediate table to table list if not already there and iterate until no join tables are needed and all join statements are determined.
- Create the FROM clause based on the table list.
- Create the WHERE clause based on the join lists and initial WHERE clause.
- Return a fully qualified query to the user to then be sent to the database (alternately do the query for the user and return the results).

This algorithm allows us to find joins simply and quickly and allows intermediate joins to tables as well. StarView uses this algorithm in a simple Perl CGI script. Its use has both improved the performance of the program and diminished the amount of support needed to update the system.

References

- Kennedy, B. & Mayhew, B. 1999, in ASP Conf. Ser., Vol. 172, *Astronomical Data Analysis Software and Systems VIII*, ed. D. M. Mehringer, R. L. Plante, & D. A. Roberts (San Francisco: ASP), 383
- Williams, J. 1994, in ASP Conf. Ser., Vol. 61, *Astronomical Data Analysis Software and Systems III*, ed. D. R. Crabtree, R. J. Hanisch, & J. Barnes (San Francisco: ASP), 96