Publication number ESL-TR-03/01-03

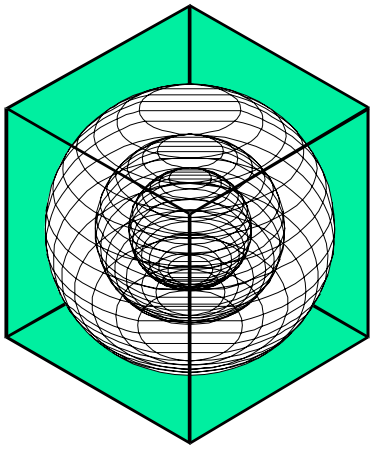
LOANSTAR DATABASE OVERVIEW

A LoanSTAR Deliverable Report

Final Report

James Sweeney, Jr.
Senior Research Associate

January 2003



ENERGY SYSTEMS LABORATORY

Texas Engineering Experiment Station
Texas A&M University System

LoanSTAR Database Overview

Energy Systems Lab
Texas A&M University

System Characteristics:
The energy data system being transferred to the Texas State Energy Conservation Office is based on the following information technologies:

Hardware: IBM eServer, X Series 240, Pentium III 1.0 GHz
Operating System Distribution: Red Hat 7.3 (Valhalla)
Operating System Kernel: Red Hat Kernel 2.4.18-3
Relational Database Management System: mySQL, Version 3.23.49
Root (system) password – r2tsn02l
Root mySQL password - n3tnrgy

General Info:
The data in this database is the LoanSTAR segment of the building energy information data system at the Texas A&M University's Energy Systems Lab. This portion of the data system has been exported to the afore-mentioned computer system in a mySQL database called lsd (LoanSTAR database). This database consists of control and channel data tables. Control data is all the data that provides relationships to the other tables in the database. The channel data is the 'energy' data.

A channel is an abstraction to a physical point of measurement. For example, a Motor Control Center Phase A is one point of measurement. There would be a single channel table for this point. This channel data is related to site and logger information by way of the control tables. The control tables provide the relational context of all the energy data.

The control tables (and descriptions) are as follows:

chids table:

The chids table is the table that contains a listing of all the channel identifications (chids) in the database. This table relates the channels to each site. Description, integration period, and other data are also available in the table.

- ch_id – integer, not null,
- site – smallint, not null,
- description - char(40), not null,
- int_period - smallint, not null,
- sum_or_avg - char(1),
- units - varchar(30),
- server - char(1) not null

comments table:

The comments table contains information specifying specific comments about certain sites.

- site – smallint, not null,
- date - varchar(20), not null,
- channel - smallint,
- mycomment - text - not null,
- who - char(10), not null

facility table:

The facility table contains information that specifies data for the facility itself, like name, contact name and title, addresses, and other facility oriented information.

- facility_id – smallint, not null,
- fac_name - varchar(50),
- contact_name - varchar(40),
- contact_title - varchar(40),
- dept - varchar(40),
- addr1 - varchar(40),
- addr2 - varchar(40),
- addr3 - varchar(40),
- contact_phone - varchar(20),
- contact_fax - varchar(20),
- other1 - varchar(80),
- other2 - varchar(80),
- user_name - char(18),
- run_time – datetime, not null, default '0000-00-00 00:00:00'

logger table:

The logger table contains data that relates the logger identification number with important details about the logger. The most important data in this table are the logger_serial_no, start_time and the comments. The logger_serial_no is the logger's serial number assigned by the manufacturer. The start_time field specifies the logger start time for that particular logger_serial_no and parm_set_label. The parm_set_label represents the parameter set label, which is an alphabetic character attached to the logger serial number for identification. Parameter sets change over time and can be distinguished from each other via this parameter set label. The comments field describes the logger in simple terms.

- logger_id - smallint, not null,
- logger_name - char(128),

- project_id - smallint,
- start_time – datetime, not null, default '0000-00-00 00:00:00',
- logger_serial_no – smallint, not null,
- parm_set_label - char(1), not null,
- comments - varchar(255),
- run_time - datetime not null, default '0000-00-00 00:00:00'

mark_action table:

The mark_action table is a table that lists the marks (or corrections) done to channel data. This tables combined with the mark_chid table, gives all the information about a correction that has occurred on a certain channel. This particular data tables stores the text comments and references therein.

- action_id integer not null,
- stmt_id smallint not null,
- seqno smallint not null,
- stmttext varchar(65),
- run_time datetime not null default '0000-00-00 00:00:00'

mark_chid table:

The mark_chid table associates a channel id (ch_id) with its marking characteristics. The begin and end times, descriptions, and mark action references are present. Combined with the mark_action table, this table provides a reference to the corrections that have been made to specific channels.

- ch_id – integer, not null,
- begin_time – datetime, not null, default '0000-00-00 00:00:00',
- end_time - datetime, not null, default '0000-00-00 00:00:00',
- description - varchar(255),
- action_id - integer, not null,
- run_time – datetime, not null default '0000-00-00 00:00:00'

notlogger table:

The notlogger data specifies logger data for data sources that are not of the Synergistic logger type. These could include, but are not limited to, other logger types such as Square D variants and software sources such as other databases.

- notlogger_id – smallint,not null,
- project_id - smallint, not null,
- name - char(160), not null

out_of_range table:

The out_of_range table is a table of channel ids and times that were out of range based on the highs and lows stored in the rawmap table. The value that was out of range is also stored in this table.

- ch_id - smallint,
- timestamp – datetime, not null, default '0000-00-00 00:00:00',
- value - float(30,15)

rawmap table:

The rawmap table stores important information about the channel ids. The channel ids are stored with start_times and logger identification (logger_id). The raw_cp field is the "channel position", or what position a channel was in the original source data file. High and low data fields are also present in this table. This provides high and low filtering of the data if necessary.

- ch_id - integer, not null,
- start_time - datetime, not null, default '0000-00-00 00:00:00',
- logger_id - smallint,not null,
- raw_cp - smallint, not null,
- high - float(30,15),
- low - float(30,15),
- run_time – datetime, not null, default '0000-00-00 00:00:00'

site_info table:

The site_info table stores the site information for each site. A site can be a collection of loggers or it may be just one logger. This table has information pertaining to each individual site. One of the most useful attributes of the table is the site_name field.

- site – smallint, not null,
- site_name - varchar(50), not null,
- site_id - char(3), not null,
- monitor_start_date - varchar(20),
- facility_id - smallint,
- mecr_id - char(10),
- loan_num - char(16),
- loan_count - char(5),
- user_name - char(18),
- run_time – datetime, not null, default '0000-00-00 00:00:00'

site_expr table:

The site_expr (site expression) table contains data about each site's whole building expressions. Whole building expressions are formulas for certain types of whole building energy types. Examples of whole building energy types are 'wbele' for whole building electrical, 'wbheat' for whole building heating and 'wbcool' for whole building cooling. There are other expressions as well, such as for chillers and control centers.

- site – smallint, not null,
- myorder – smallint, not null,
- name - varchar(20), not null,
- expr - varchar(250), not null,
- user_name - char(18),
- run_time – datetime, not null, default '0000-00-00 00:00:00'

site_energy table:

The site_energy table contains data stores important information relative to the calculation of savings. One of the most important parameters stored here is the area of the building.

- site – smallint, not null,
- energy_type - varchar(20), not null,
- const_start_date - varchar(20),
- const_end_date - varchar(20),
- area - float(30,15),
- user_name - char(18),
- run_time - datetime, not null, default '0000-00-00 00:00:00'

All other tables in this database are channel tables. Channel tables are of the format "ch####", where #### is a sequence of numbers.

<u>Using mySQL (a brief guide):</u>
A more detailed guide is included in the CD.

Root (system) password – r2tsn02l
Root mysql password - n3tnrgy

You must be logged in as mysql root (AKA mysql administrator) to perform high level functions. To use mysqladmin use the following:
>mysqladmin -u root -p 'command'

Logging in as mysql root:
>mysql -u root -p
enter password

To see what databases are in the engine:

mysql>show DATABASES;

To select a database:
mysql>use ' testdatabase'

To see what tables are in database:
mysql>show tables;

To see the meta contents of a table:
mysql>describe 'table-name';

To select data from a table is just like normal SQL
mysql>select * from 'table-name';

To pipe SQL to a db
>mysql -u root -p -D [database name] < mysqlfile.sql

Or from the Command line:

To enable LOAD DATA LOCAL (to load data from a local file):
>mysql -u root -p –local-infile=1

You must be logged in as mysql root to perform high level functions.
Also to use mysqladmin use the following:
>mysqladmin -u root -p 'command'

Logging in as mysql root:
>mysql -u root -p
enter password

To enable LOAD DATA LOCAL:
>mysql -u root -p --local-infile=1

To see what databases are in the engine:
>show DATABASES;

To select a database:
>use testdatabase

To see what tables are in database:
>show tables;

To see the meta contents of a table:
>describe 'table-name';

To pipe SQL to a db
>mysql -u root -p -D [database name] < mysqlfile.sql

Tools for Security Checking
---
nmap
shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings

Setting up new users

These examples assume that privileges are set up according to the defaults described in the previous section. This means that to make changes, you must be on the same machine where mysqld is running, you must connect as the MySQL root user, and the root user must have the INSERT privilege for the mysql database and the RELOAD administrative privilege. Also, if you have changed the root user password, you must specify it for the mysql commands here.

You can add new users by issuing GRANT statements:

```
shell> mysql --user=root mysql
mysql> GRANT ALL PRIVILEGES ON *.* TO monty@localhost
    ->     IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT ALL PRIVILEGES ON *.* TO monty@"%"
    ->     IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT RELOAD,PROCESS ON *.* TO admin@localhost;
mysql> GRANT USAGE ON *.* TO dummy@localhost;
```

These GRANT statements set up three new users:

monty
A full superuser who can connect to the server from anywhere, but who must use a password 'some_pass' to do so. Note that we must issue GRANT statements for both monty@localhost and monty@"%". If we don't add the entry with localhost, the anonymous user entry for localhost that is created by mysql_install_db will take precedence when we connect from the local host, because it has a more specific Host field value and thus comes earlier in the user table sort order.

admin
A user who can connect from localhost without a password and who is granted the RELOAD and PROCESS administrative privileges. This allows the user to execute the mysqladmin reload, mysqladmin refresh, and mysqladmin flush-* commands, as well as mysqladmin processlist . No database-related privileges are granted. (They can be granted later by issuing additional GRANT statements.)

dummy
A user who can connect without a password, but only from the local host. The global privileges are all set to 'N'�the USAGE privilege type allows you to create a user with no privileges. It is assumed that you will grant database-specific privileges later.

Creating a Table

Creating the database is the easy part, but at this point it's empty, as SHOW TABLES will tell you:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

The harder part is deciding what the structure of your database should be: what tables you will need and what columns will be in each of them.

You'll want a table that contains a record for each of your pets. This can be called the pet table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example,

if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it's not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it's better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you'll soon need to send out birthday greetings, for that computer-assisted personal touch.)
You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died. You can probably think of other types of information that would be useful in the pet table, but the ones identified so far are sufficient for now: name, owner, species, sex, birth, and death.

Use a CREATE TABLE statement to specify the layout of your table:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
    -> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

VARCHAR is a good choice for the name, owner, and species columns because the column values will vary in length. The lengths of those columns need not all be the same, and need not be 20. You can pick any length from 1 to 255, whatever seems most reasonable to you. (If you make a poor choice and it turns out later that you need a longer field, MySQL provides an ALTER TABLE statement.)

Animal sex can be represented in a variety of ways, for example, "m" and "f", or perhaps "male" and "female". It's simplest to use the single characters "m" and "f".

The use of the DATE data type for the birth and death columns is a fairly obvious choice.

Now that you have created a table, SHOW TABLES should produce some output:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| pet                 |
+---------------------+
```

To verify that your table was created the way you expected, use a DESCRIBE statement:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

You can use DESCRIBE any time, for example, if you forget the names of the columns in your table or what types they are.

Loading Data into a Table

After creating your table, you need to populate it. The LOAD DATA and INSERT statements are useful for this.

Suppose your pet records can be described as shown here. (Observe that MySQL expects dates in YYYY-MM-DD format; this may be different from what you are used to.)

```
name  owner  species  sex  birth  death
Fluffy  Harold  cat  f  1993-02-04
Claws  Gwen  cat  m  1994-03-17
Buffy  Harold  dog  f  1989-05-13
Fang  Benny  dog  m  1990-08-27
Bowser  Diane  dog  m  1998-08-31  1995-07-29
Chirpy  Gwen  bird  f  1998-09-11
Whistler  Gwen  bird   1997-12-09
Slim  Benny  snake  m  1996-04-29
```

Because you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your animals, then load the contents of the file into the table with a single statement.

You could create a text file `pet.txt' containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the CREATE TABLE statement. For missing values (such as unknown sexes or death dates for animals that are still living), you can use NULL values. To represent these in your text file, use \N. For example, the record for Whistler the bird would look like this (where the whitespace

between values is a single tab character):

```
name  owner  species  sex  birth  death
Whistler  Gwen  bird  \N  1997-12-09  \N
```

To load the text file `pet.txt' into the pet table, use this command:

```
mysql> LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
```

You can specify the column value separator and end of line marker explicitly in the LOAD DATA statement if you wish, but the defaults are tab and linefeed. These are sufficient for the statement to read the file `pet.txt' properly.

When you want to add new records one at a time, the INSERT statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the CREATE TABLE statement. Suppose Diane gets a new hamster named Puffball. You could add a new record using an INSERT statement like this:

```
mysql> INSERT INTO pet
    -> VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

Note that string and date values are specified as quoted strings here. Also, with INSERT, you can insert NULL directly to represent a missing value. You do not use \N like you do with LOAD DATA.

From this example, you should be able to see that there would be a lot more typing involved to load your records initially using several INSERT statements rather than a single LOAD DATA statement.