dbsake Documentation

Release 2.1.2

Andrew Garner

Jan 30, 2019

Contents

dbsake is a collection of command-line tools to perform various DBA related tasks for MySQL.

```
# curl -s http://get.dbsake.net > dbsake
# chmod u+x dbsake
# dbsake sandbox
```

CHAPTER 1

dbsake

dbsake - a (s)wiss-(a)rmy-(k)nif(e) for MySQL

- Free software: GPLv2
- Documentation: http://docs.dbsake.net.

1.1 Features

- Parsing MySQL .frm files and output DDL
- Filtering and transforming mysqldump output
- Patching a my.cnf to remove or convert deprecated options
- Deploying a new standalone MySQL "sandbox" instance
- Decoding/encoding MySQL filenames
- Managing OS caching for a set of files

1.2 Dependencies

- Requires python v2.6+
- jinja2 >= 2.2
- click >= 2.0

1.3 Reporting Bugs

If you find a bug in dbsake please report the issue on the dbsake issue on github

If you know how to fix the problem feel free to fork dbsake and submit a pull request. See Contributing for more information.

1.4 Quickstart

You can fetch dbsake easily from get.dbsake.net:

```
$ curl -s get.dbsake.net > dbsake
```

This is an executable python zip archive with all dependencies included.

You can run as a script by making it executable:

```
$ chmod u+x dbsake
```

Run it with no arguments to see all possible commands:

```
$ dbsake
Usage: dbsake [options] <command>

Options:
  -d, --debug
  -q, --quiet
  -V, --version  Show the version and exit.
  -?, --help     Show this message and exit.

Commands:
  decode-tablename  Decode a MySQL filename.
  encode-tablename  Encode a MySQL table identifier.
  fincore           Report cached pages for a file.
  frmdump           Dump schema from MySQL frm files.
  help              Show help for a command.
  sandbox           Create a sandboxed MySQL instance.
  sieve             Filter and transform mysqldump output.
  uncache           Uncache file(s) from the OS page cache.
  upgrade-mycnf     Upgrade a MySQL option file.
```

1.4.1 "Upgrading" a my.cnf

Here's how you might upgrade a MySQL 5.0 my.cnf to 5.5:

```
$ dbsake upgrade-mycnf --target=5.5 --config=my.cnf --patch
Rewriting option 'log-slow-queries'. Reason: Logging options changed in MySQL 5.1
Removing option 'skip-external-locking'. Reason: Default behavior in MySQL 4.1+
--- a/my.cnf
+++ b/my.cnf
@@ -26,7 +26,6 @@
 [mysqld]
 port        = 3306
 socket      = /var/run/mysqld/mysqld.sock
-skip-external-locking
 key_buffer_size = 384M
 max_allowed_packet = 1M
 table_open_cache = 512
```

(continues on next page)

(continued from previous page)

```
@@ -127,7 +126,9 @@
 #innodb_log_buffer_size = 8M
 #innodb_flush_log_at_trx_commit = 1
 #innodb_lock_wait_timeout = 50
-log-slow-queries = /var/lib/mysql/slow.log
+slow-query-log = 1
+slow-query-log-file = /var/lib/mysql/slow.log
+log-slow-slave-statements

 [mysqldump]
 quick
```

1.4.2 Processing mysqldump output

Here's how you filter a single table from a mysqldump:

```
$ mysqldump -A | dbsake sieve --to-stdout -t mysql.db
-- MySQL dump 10.14  Distrib 5.5.38-MariaDB, for Linux (x86_64)
--
-- Host: localhost    Database:
-- ------------------------------------------------------
-- Server version   5.5.38-MariaDB-log

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `db`
--

DROP TABLE IF EXISTS `db`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `db` (
  `Host` char(60) COLLATE utf8_bin NOT NULL DEFAULT '',
  `Db` char(64) COLLATE utf8_bin NOT NULL DEFAULT '',
  `User` char(16) COLLATE utf8_bin NOT NULL DEFAULT '',
  `Select_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Insert_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Update_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Delete_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Create_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Drop_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Grant_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `References_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Index_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Alter_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
```

(continues on next page)

(continued from previous page)

```
  `Create_tmp_table_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Lock_tables_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Create_view_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Show_view_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Create_routine_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Alter_routine_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Execute_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Event_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  `Trigger_priv` enum('N','Y') CHARACTER SET utf8 NOT NULL DEFAULT 'N',
  PRIMARY KEY (`Host`,`Db`,`User`),
  KEY `User` (`User`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_bin COMMENT='Database privileges';
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `db`
--

LOCK TABLES `db` WRITE;
/*!40000 ALTER TABLE `db` DISABLE KEYS */;
/*!40000 ALTER TABLE `db` ENABLE KEYS */;
UNLOCK TABLES;

/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2014-07-22 21:01:35
```

1.4.3 Deploying a MySQL sandbox instance

Here is how you create a MySQL 5.7.3-m13 instance:

```
$ dbsake sandbox -m 5.7.3-m13
Preparing sandbox instance: /home/localuser/sandboxes/sandbox_20140722_210338
  Creating sandbox directories
    * Created directories in 0.00 seconds
  Deploying MySQL distribution
    - Deploying MySQL 5.7.3-m13 from download
    - Using cached download /home/localuser/.dbsake/cache/mysql-5.7.3-m13-linux-
↪glibc2.5-x86_64.tar.gz
    - Verifying gpg signature via: /usr/bin/gpg2 --verify /home/localuser/.dbsake/
↪cache/mysql-5.7.3-m13-linux-glibc2.5-x86_64.tar.gz.asc -
    - Unpacking tar stream. This may take some time
(100.00%)[=======================================] 322.9MiB / 322.9MiB
    - GPG signature validated
    * Deployed MySQL distribution in 13.56 seconds
  Generating my.sandbox.cnf
    - Generated random password for sandbox user root@localhost
```

(continues on next page)

(continued from previous page)

```
    * Generated /home/localuser/sandboxes/sandbox_20140722_210338/my.sandbox.cnf in 0.
↪03 seconds
  Bootstrapping sandbox instance
    - Logging bootstrap output to /home/localuser/sandboxes/sandbox_20140722_210338/
↪bootstrap.log
    * Bootstrapped sandbox in 2.67 seconds
  Creating sandbox.sh initscript
    * Generated initscript in 0.01 seconds
Sandbox created in 16.28 seconds

Here are some useful sandbox commands:
       Start sandbox: /home/localuser/sandboxes/sandbox_20140722_210338/sandbox.sh␣
↪start
        Stop sandbox: /home/localuser/sandboxes/sandbox_20140722_210338/sandbox.sh␣
↪stop
  Connect to sandbox: /home/localuser/sandboxes/sandbox_20140722_210338/sandbox.sh␣
↪mysql <options>
   mysqldump sandbox: /home/localuser/sandboxes/sandbox_20140722_210338/sandbox.sh␣
↪mysqldump <options>
Install SysV service: /home/localuser/sandboxes/sandbox_20140722_210338/sandbox.sh␣
↪install-service
```

The sandbox.sh script has some convenient commands for interacting with the sandbox too:

```
$ /home/localuser/sandboxes/sandbox_20140722_210338/sandbox.sh start
Starting sandbox: .[OK]

$ /home/localuser/sandboxes/sandbox_20140722_210338/sandbox.sh mysql -e 'select␣
↪@@datadir, @@version, @@version_comment\G'
*************************** 1. row ***************************
        @@datadir: /home/localuser/sandboxes/sandbox_20140722_210338/data/
        @@version: 5.7.3-m13-log
@@version_comment: MySQL Community Server (GPL)
```

The sandbox.sh script can also install itself, if you want to make the sandbox persistent:

```
$ sudo /home/localuser/sandboxes/sandbox_20140722_210338/sandbox.sh install-service
+ /bin/cp /home/localuser/sandboxes/sandbox_20140722_210338/sandbox.sh /etc/init.d/
↪mysql-5.7.3
+ /sbin/chkconfig --add mysql-5.7.3 && /sbin/chkconfig mysql-5.7.3 on
Service installed in /etc/init.d/mysql-5.7.3 and added to default runlevels
```

1.4.4 Dumping the schema from MySQL .frm files

Here's an example dumping a normal table's .frm:

```
$ sudo dbsake frmdump /var/lib/mysql/sakila/actor.frm
--
-- Table structure for table `actor`
-- Created with MySQL Version 5.5.34
--

CREATE TABLE `actor` (
  `actor_id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
  `first_name` varchar(45) NOT NULL,
```

(continues on next page)

(continued from previous page)

```
  `last_name` varchar(45) NOT NULL,
  `last_update` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_
↪TIMESTAMP,
  PRIMARY KEY (`actor_id`),
  KEY `idx_actor_last_name` (`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

You can also format VIEW .frm files directly as well:

```
$ sudo dbsake frmdump /var/lib/mysql/sakila/actor_info.frm
--
-- View:         actor_info
-- Timestamp:    2014-01-18 18:22:54
-- Stored MD5:   402b8673b0c61034644b5b286519d3f1
-- Computed MD5: 402b8673b0c61034644b5b286519d3f1
--

CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY INVOKER VIEW␣
↪`actor_info` AS select `a`.`actor_id` AS `actor_id`,`a`.`first_name` AS `first_
↪name`,`a`.`last_name` AS `last_name`,group_concat(distinct concat(`c`.`name`,': ',
↪(select group_concat(`f`.`title` order by `f`.`title` ASC separator ', ') from␣
↪((`sakila`.`film` `f` join `sakila`.`film_category` `fc` on((`f`.`film_id` = `fc`.
↪`film_id`))) join `sakila`.`film_actor` `fa` on((`f`.`film_id` = `fa`.`film_id`)))␣
↪where ((`fc`.`category_id` = `c`.`category_id`) and (`fa`.`actor_id` = `a`.`actor_
↪id`)))) order by `c`.`name` ASC separator '; ') AS `film_info` from (((`sakila`.
↪`actor` `a` left join `sakila`.`film_actor` `fa` on((`a`.`actor_id` = `fa`.`actor_
↪id`))) left join `sakila`.`film_category` `fc` on((`fa`.`film_id` = `fc`.`film_
↪id`))) left join `sakila`.`category` `c` on((`fc`.`category_id` = `c`.`category_
↪id`))) group by `a`.`actor_id`,`a`.`first_name`,`a`.`last_name`;
```

CHAPTER 2

Usage

2.1 Synopsis

dbsake [-d|–debug] [-q|–quiet] [-V|–version] [-?|–help] <command> [<args>]

2.2 Description

dbsake is a collection of command-line tools to assist with administering parts of a MySQL database.

Formatted and hyperlinked versions of the latest dbsake documentation can be found at http://docs.dbsake.net.

2.3 Options

-?, --help
Show the top-level dbsake options

-V, --version
Output the current dbsake version and exit

-q, --quiet
Suppresses all logging output. Commands that output to stdout will still emit output, but no logging will be performed. You can use the exit status of dbsake to detect failure in these cases

-d, --debug
Enable debugging output. This enables more verbose logs that are typically not necessary, but may be helpful for troubleshooting.

2.4 Enabling Bash completion

dbsake supports bash completion via the python-click library. To enable this you can run the following command:

```
eval $(_DBSAKE_COMPLETE=source dbsake)
```

This will enable tab completing subcommands and options.

Bash completion only works if the script is run directly and not directly via the python interpreter. That is the following will use tab completion:

```
# eval $(_DBSAKE_COMPLETE=source ./dbsake)
# ./dbsake <TAB>
```

However, the following command will not:

```
# eval $(_DBSAKE_COMPLETE=source dbsake)
# python dbsake<TAB>
```

CHAPTER 3

Commands

3.1 decode-tablename

Decode a MySQL encoded filename

As of MySQL 5.1, tablenames with special characters are encoded with a custom "filename" encoding. This command reverses that process to output the original tablename.

3.1.1 Usage

```
Usage: dbsake decode-tablename [options] [NAMES]...

  Decode a MySQL tablename as a unicode name.

Options:
  -?, --help  Show this message and exit.
```

3.1.2 Example

```
$ dbsake decode-tablename foo@002ebar
foo.bar
```

3.1.3 Options

path [path...]
: Specify a filename to convert to plain unicode

3.2 encode-tablename

Encode a MySQL tablename with the MySQL filename encoding

This is the opposite of filename-to-tablename, where it takes a normal tablename and converts it using MySQL's filename encoding.

3.2.1 Usage

```
Usage: dbsake encode-tablename [options] [NAMES]...

  Encode a MySQL tablename

Options:
  -?, --help  Show this message and exit.
```

3.2.2 Example

```
$ dbsake encode-tablename foo.bar
foo@002ebar
```

3.2.3 Options

path [path...]
    Specify a tablename to convert to an encoded filename

3.3 frmdump

Decode a MySQL .frm file and output a CREATE VIEW or CREATE TABLE statement.

This command does not require a MySQL server and interprets a .frm file according to rules similar to the MySQL server.

For more information on how this command works see *Description of the .frm format*

Important: This program only decodes data strictly available in the .frm file. InnoDB foreign-key references are not preserved and AUTO_INCREMENT values are also not preserved as these are stored outside of the .frm.

3.3.1 Usage

```
Usage: dbsake frmdump [options] [path...]

  Dump schema from MySQL frm files.

Options:
  -t, --type-codes  Show mysql type codes in comments on each column
  -r, --recursive   Recursively search directories for .frm files.
```

(continues on next page)

(continued from previous page)

```
  -R, --replace     Output views with CREATE OR REPLACE
  -?, --help        Show this message and exit.
```

3.3.2 Example

```
$ dbsake frmdump --type-codes /var/lib/mysql/mysql/plugin.frm
--
-- Table structure for table `plugin`
-- Created with MySQL Version 5.5.35
--

CREATE TABLE `plugin` (
  `name` varchar(64) NOT NULL DEFAULT ''  /* MYSQL_TYPE_VARCHAR */,
  `dl` varchar(128) NOT NULL DEFAULT ''  /* MYSQL_TYPE_VARCHAR */,
  PRIMARY KEY (`name`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COMMENT='MySQL plugins';


$ dbsake frmdump /var/lib/mysql/sakila/actor_info.frm
--
-- View:         actor_info
-- Timestamp:    2014-01-04 05:29:55
-- Stored MD5:   402b8673b0c61034644b5b286519d3f1
-- Computed MD5: 402b8673b0c61034644b5b286519d3f1
--

CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY INVOKER VIEW␣
↪`actor_info` select `a`.`actor_id` AS `actor_id`,`a`.`first_name` AS `first_name`,
↪`a`.`last_name` AS `last_name`,group_concat(distinct concat(`c`.`name`,': ',(select␣
↪group_concat(`f`.`title` order by `f`.`title` ASC separator ', ') from ((`sakila`.
↪`film` `f` join `sakila`.`film_category` `fc` on((`f`.`film_id` = `fc`.`film_id`)))␣
↪join `sakila`.`film_actor` `fa` on((`f`.`film_id` = `fa`.`film_id`))) where ((`fc`.
↪`category_id` = `c`.`category_id`) and (`fa`.`actor_id` = `a`.`actor_id`)))) order␣
↪by `c`.`name` ASC separator '; ') AS `film_info` from (((`sakila`.`actor` `a` left␣
↪join `sakila`.`film_actor` `fa` on((`a`.`actor_id` = `fa`.`actor_id`))) left join␣
↪`sakila`.`film_category` `fc` on((`fa`.`film_id` = `fc`.`film_id`))) left join␣
↪`sakila`.`category` `c` on((`fc`.`category_id` = `c`.`category_id`))) group by `a`.
↪`actor_id`,`a`.`first_name`,`a`.`last_name`;
```

3.3.3 Options

Changed in version 2.0.0: frm-to-schema was renamed to frmdump

-R, --replace
Output view as CREATE OR REPLACE so that running the DDL against MySQL will overwrite a view.

-t, --type-codes
Add comment to base tables noting the underlying mysql type code as MYSQL_TYPE_<name>.

-r, --recursive
If any directory path is specified on the commandline, recursively search that directory and dump any files ending in .frm

path [path...]
Specify the .frm files to generate a CREATE TABLE command from.

New in version 1.0.2: Support for indexes with a prefix length in binary .frm files; e.g. KEY (blob_value(255))

Changed in version 1.0.2: Views are parsed from .frm files rather than skipped.

Changed in version 1.0.2: Raw MySQL types are no longer added as comments unless the –raw-types option is specified.

Changed in version 1.0.2: A – Table structure for table '<name>' comment is added before each table

Changed in version 2.0.0: The `--raw-types` option was renamed to *`frmdump --type-codes`*.

New in version 1.0.2: The *`frmdump --replace`* option

New in version 2.1.1: The *`frmdump --recursive`* option

3.4 sandbox

New in version 1.0.3.

Setup a secondary MySQL instance painlessly.

This setups a MySQL under ~/sandboxes/ (by default) with a randomly generated password for the root@localhost user and networking disabled.

A simple shell script is provided to start, stop and connect to the MySQL instance.

Changed in version 1.0.5: dbsake verifies the gpg signature of downloaded MySQL tarball distributions

---

Important: As of dbsake 2.0.0, the sandbox options have changed. -D is now an alias for –datadir, although it was previously an alias for –data-source. The -s option is now an alias for –data-source in order to specify a tarball to seed the sandbox instance with. See *`sandbox --datadir`* and *`sandbox --data-source`* for more information.

3.4.1 Usage

```
Usage: dbsake sandbox [OPTIONS]

  Create a sandboxed MySQL instance.

  This command installs a new MySQL instance under the specified sandbox
  directory, or under ~/sandboxes/sandbox_<datetime> if none is specified.

Options:
  -d, --sandbox-directory <path>  path where sandbox will be installed
  -m, --mysql-distribution <dist>
                                  mysql distribution to install
  -D, --datadir <path>            Path to datadir for sandbox
  -s, --data-source <source>      path to file to populate sandbox
  -t, --table <glob-pattern>      db.table glob pattern to include from
                                  --data-source
  -T, --exclude-table <glob-pattern>
                                  db.table glob pattern to exclude from
                                  --data-source
  -c, --cache-policy <policy>     cache policy to apply when downloading mysql
                                  distribution
  --skip-libcheck                 skip check for required system libraries
  --skip-gpgcheck                 skip gpg verification of download mysql
```

(continues on next page)

(continued from previous page)

```
                                distributions
  --force                       overwrite existing sandbox directory
  -u, --mysql-user <user>       MySQL user to add to the sandbox instance
  -p, --password                prompt for password to create root@localhost
                                with
  -x, --innobackupex-options <options>
                                additional options to run innobackupex
                                --apply-logs
  -?, --help                    Show this message and exit.
```

3.4.2 Example

```
# dbsake sandbox --sandbox-directory=/opt/mysql-5.6.19 \
>                --mysql-distribution=5.6.19 \
>                --data-source=backup.tar.gz
Preparing sandbox instance: /opt/mysql-5.6.19
  Creating sandbox directories
    * Created directories in 0.00 seconds
  Preloading sandbox data from /root/backup.tar.gz
(100.00%)[========================================] 276.0KiB / 276.0KiB
    - Sandbox data appears to be unprepared xtrabackup data
    - Running: /root/xb/bin/innobackupex --apply-log  .
    - (cwd: /opt/mysql-5.6.19/data)
    - innobackupex --apply-log succeeded. datadir is ready.
    * Data extracted in 4.46 seconds
  Deploying MySQL distribution
    - Deploying MySQL 5.6.19 from download
    - Downloading from http://cdn.mysql.com/Downloads/MySQL-5.6/mysql-5.6.19-linux-
↪glibc2.5-x86_64.tar.gz
    - Importing mysql public key to /root/.dbsake/gpg
    - Verifying gpg signature via: /bin/gpg2 --verify /root/.dbsake/cache/mysql-5.6.
↪19-linux-glibc2.5-x86_64.tar.gz.asc -
    - Unpacking tar stream. This may take some time
(100.00%)[========================================] 291.4MiB / 291.4MiB
    - GPG signature validated
    - Stored MD5 checksum for download: /root/.dbsake/cache/mysql-5.6.19-linux-glibc2.
↪5-x86_64.tar.gz.md5
    * Deployed MySQL distribution in 46.17 seconds
  Generating my.sandbox.cnf
    - Generated random password for sandbox user root@localhost
    ! Existing ib_logfile0 detected. Setting innodb-log-file-size=5M
    ! Found existing shared innodb tablespace: ibdata1:18M:autoextend
    * Generated /opt/mysql-5.6.19/my.sandbox.cnf in 0.03 seconds
  Bootstrapping sandbox instance
    - Logging bootstrap output to /opt/mysql-5.6.19/bootstrap.log
    - User supplied mysql.user table detected.
    - Skipping normal load of system table data
    - Ensuring root@localhost exists
    * Bootstrapped sandbox in 2.04 seconds
  Creating sandbox.sh initscript
    * Generated initscript in 0.01 seconds
Sandbox created in 52.72 seconds

Here are some useful sandbox commands:
       Start sandbox: /opt/mysql-5.6.19/sandbox.sh start
```

(continues on next page)

(continued from previous page)

```
       Stop sandbox: /opt/mysql-5.6.19/sandbox.sh stop
  Connect to sandbox: /opt/mysql-5.6.19/sandbox.sh mysql <options>
   mysqldump sandbox: /opt/mysql-5.6.19/sandbox.sh mysqldump <options>
Install SysV service: /opt/mysql-5.6.19/sandbox.sh install-service
```

3.4.3 Options

Changed in version 2.0.0: mysql-sandbox renamed to sandbox

-d, --sandbox-directory <path>
Specify the path under which to create the sandbox. This defaults to ~/sandboxes/sandbox_$(date +%Y%m%d_%H%M%S)

Changed in version 1.0.6: –sandbox-directory supports relative paths

-m, --mysql-distribution <name>
Specify the source for the mysql distribution. This can be one of:

- **system - use the local mysqld binaries already installed on** the system
- mysql*.tar.gz - path to a tarball distribution
- **<mysql-version> - if a mysql version is specified then an** attempt is made to download a binary tarball from dev.mysql.com and otherwise is identical to installing from a local tarball

The default, if no option is specified, will be to use system which copies the minimum binaries from system director to $sandbox_directory/bin/.

Changed in version 1.0.4: –mysql-source was renamed to –mysql-distribution

---

Note: –mysql-distribution = <version> will only auto-download tarballs from mysql.com. To install Percona or MariaDB sandboxes, you will need to download the tarballs separately and specify the tarball path via –mysql-distribution /path/to/my/tarball

---

-D, --datadir <path>
Specify the path to the datadir to be used for the sandbox. If this path does not exist, it will be created. The datadir will be boostrapped using the MySQL version specified via the *sandbox --mysql-distribution* option. Sanity checks will be done against the path to verify that it is either empty or seems to be a valid, unused MySQL datadir.

New in version 2.0.0.

-s, --data-source <tarball>
Specify a tarball that will be used for the sandbox datadir. If a tarball is specified it will be extracted to the ./data/ path under the sandbox directory, subject to any filtering specified by the –table and –exclude-table options.

New in version 1.0.4.

Changed in version 2.0.0: The `-s` short option was added. In 1.0 this was `-D`, but as of 2.0.0, -D is an alias for –datadir.

Changed in version 2.0.0: –data-source now only takes a tarball option. To use an existing datadir, use the *sandbox --datadir* option.

Changed in version 1.0.5: A directory may be specified for the –data-source option to use an existing datadir for the sandbox.

Note: Support for tarballs in –data-source is presently limited to tarballs relative to the datadir - such as those generated by percona-xtrabackup or certain LVM snapshot backup utilities.

Directory data sources have no filtering applied even if –table or –exclude-table options were provided.

-t, --table <glob>
Specify a glob pattern to filter elements from the –data-source option. If –data-source is not specified this option has no effect. <glob> should be of the form database.table with optional glob special characters. This use the python fnmatch mechanism under the hood so is limited to only the *, ?, [seq] and [!seq] glob operations.

New in version 1.0.4.

Changed in version 2.0.0: `--table="mysql.*"` is included by default in the list of table options regardless of other *sandbox --table* optons. Tables in the mysql schema can be excluded by using the *sandbox --exclude-table* option.

-T, --exclude-table <glob>
Specify a glob pattern to filter elements from the –data-source option. If –data-source is not specified this option has no effect.

New in version 1.0.4.

-c, --cache-policy <always|never|refresh|local>
Specify the cache policy if installing a MySQL distribution via a download (i.e when only a version is specified). This command will cache downloaded tarballs by default in the directory specified by $DBSAKE_CACHE environment variable, or ~/.dbsake/cache if this is not specified.

The cache policies have the following semantics:

- always - check cache and update the cache if a download is required
- never - never use the cache - this will always result in a download
- refresh - skip the cache, but update it from a download
- local - check cache, but fail if a local tarball is not present

New in version 1.0.4.

--skip-libcheck
As of dbsake 1.0.5, if a version of MySQL >= 5.5.4 is requested for download, dbsake checks for libaio on the system. Without libaio mysqld from any recent version of MySQL will fail to start at all. This option allows proceeding anyway in case, dbsake is not detecting libaio correctly. Use of this option will often cause the sandbox process to just fail later in the process.

New in version 1.0.5.

--skip-gpgcheck
Disables verification of the gpg signature when downloading MySQL tarball distributions.

New in version 1.0.5.

--force
Forces overwriting the path specified by `--sandbox-directory` if it already exists

New in version 1.0.9.

-u, --mysql-user <name>
Specify the user to add to the sandbox instance. By default this is root@localhost but can be overriden with this option to avoid changing the password for an existing root@localhost user when using the *sandbox --data-source* option.

New in version 2.0.0.

-p, --password
Prompt for the root@localhost password instead of generating a random password (the default behavior). The password will be read from stdin if this option is specified and stdin is not a TTY

New in version 1.0.9.

Changed in version 2.0.0: –prompt-password renamed to –password

-x, --innobackupex-options `<options>`
Add additional options to the "innobackupex –apply-log {extra options} ." commandline that the sandbox command uses to prepare a datadir created from an xtrabackup tarball image provided via the `--data-source` opton.

New in version 1.0.9.

3.4.4 Using the sandbox.sh control script

Usage: ./sandbox.sh <action> [options]

When creating a sandbox, mysql-sandbox generate a simple bash script to control the sandbox in ./sandbox.sh under the sandbox directory. This follows the pattern of a SysV init script and has many standard actions:

- start

  start the sandbox (noop if already started)

  Note: sandbox.sh start passes any additional options directly to the mysqld_safe script. So you can do things like: ./sandbox.sh start –init-file=reset_root.sql

- stop

  stop the sandbox (noop if already stopped)

- restart

  stop then start the sandbox

- condrestart

  only restart if sandbox is running

- status check if the sandbox is running

Additionally there are several custom actions to make managing the sandbox easier:

- metadata

  Outputs some basic information about the sandbox environment including the version, the my.cnf being used, and various mysql command paths that are used by sandbox.sh

- version

  Output a version string for the mysql server process this sandbox was initialized with.

- mysql [options]

  connect to the sandbox using the mysql command line client

  You can pass any option you might pass to mysql here. I.e: ./sanbox.sh mysql -e 'SHOW ENGINE INNODB STATUSG' For convenience the action 'use' is an alias for 'mysql'

- mysqldump [options]

  run mysqldump against the sandbox

  Example: ./sandbox.sh mysqldump –all-databases | gzip > backup.sql.gz

- upgrade [options]

  run mysql_upgrade against the sandbox

  Example: ./sandbox.sh upgrade –upgrade-system-tables

  This is useful in conjunction with the –data-source option where you might load data from a previous MySQL version into a new version for testing and want to perform an in-place upgrade of that data.

- install-service

  attempt to install the sandbox.sh under /etc/init.d and add to default runlevels. This is effectively just an alias for:

```
# cp sandbox.sh /etc/init.d/${name}
# chkconfig --add ${name} && chkconfig ${name} on

Under ubuntu update-rc.d is used instead of chkconfig.

install-service accept one argument as the name of the service to install.
By default this will be called mysql-${version} where $version is the
current mysqld version being used (e.g. 5.6.15)
```

3.5 sieve

Filter and transform a mysqldump stream.

This command processes mysqldump output, potentially filtering or transforming the output based on the provided command line options.

sieve effective works in two modes:

- streaming; mysqldump is read from `--input-file` and written to stdout possibly with different output depending on the provided options.
- directory; mysqldump is read from `--input-file` and split into separate files in the requested directory. This allows converting a large dump in a file-per-table easily. Files output in this mode are additionally filtered through `--compress-command` and are processed through `gzip --fast` by default so the output is compressed on disk by default.

3.5.1 Usage

```
Usage: dbsake sieve [options]

  Filter and transform mysqldump output.

  sieve can extract single tables from a mysqldump file and perform useful
  transformations, such as adding indexes after the table data is loaded for
  InnoDB tables, where such indexes can be created much more efficiently
  than the default incremental rebuild that mysqldump performs.
```

(continues on next page)

(continued from previous page)

```
Options:
  -F, --format <name>              Select the output format (directory, stream)
  -C, --directory <path>           Specify output directory when
                                   --format=directory
  -i, --input-file <path>          Specify input file to process instead of
                                   stdin
  -z, --compress-command <name>    Specify compression command when
                                   --format=directory
  -t, --table <glob>               Only output tables matching the given glob
                                   pattern
  -T, --exclude-table <glob>       Excludes tables matching the given glob
                                   pattern
  --defer-indexes                  Add secondary indexes after loading table
                                   data
  --defer-foreign-keys             Add foreign key constraints after loading
                                   table data
  --write-binlog / --no-write-binlog
                                   Include SQL_LOG_BIN = 0 in output to disable
                                   binlog
  --table-schema / --no-table-schema
                                   Include/exclude table schema from output.
  --table-data / --no-table-data   Include/exclude table data from output
  --routines / --no-routines       Include / exclude database routines from
                                   output
  --events / --no-events           Include / exclude database events from
                                   output
  --triggers / --no-triggers       Include/exclude table triggers from output
  --master-data / --no-master-data
                                   Uncomment/comment CHANGE MASTER in input, if
                                   present
  -O, --to-stdout                  Force output on stdout, even to a terminal.
  -?, --help                       Show this message and exit.
```

3.5.2 Example

```
$ mysqldump --routines sakila | dbsake sieve --format=directory --directory=backups/
$ tree backups
backups
└── sakila
    ├── actor.sql.gz
    ├── address.sql.gz
    ├── category.sql.gz
    ├── city.sql.gz
    ├── country.sql.gz
    ├── customer.sql.gz
    ├── film_actor.sql.gz
    ├── film_category.sql.gz
    ├── film.sql.gz
    ├── film_text.sql.gz
    ├── inventory.sql.gz
    ├── language.sql.gz
    ├── payment.sql.gz
    ├── rental.sql.gz
    ├── routines.ddl.gz
    ├── staff.sql.gz
```

(continues on next page)

(continued from previous page)

```
    ├── store.sql.gz
    └── views.ddl.gz

1 directory, 18 files
```

3.5.3 Options

Changed in version 2.0.0: Renamed split-mysqldump to sieve; Significant rewrite of functionality.

Changed in version 2.0.0: Remove –regex option in favor of -t/–table and -T/–exclude-table option which accepts globs.

-F, --format <name>
Output file format. Must be one of 'stream' or 'directory'. If set to 'stream', output will be written on stdout. Unless –force is also specified the sieve command with refuse to write to a terminal.

If set to 'directory', output will be written to the path specified by the `--directory` option, with a file per table.

New in version 2.0.0.

-C, --directory <output directory>
Path where the sieve command should create output files. Ignored if `--format` is set to 'stream'. The sieve command will create this path if it does not already exist.

Defaults to '.' - the current working directory.

-i, --input-file <path>
Input file to read mysqldump input from. Default to "-" and reads from stdin. This must be an uncompressed data source, so to process an already compressed .sql.gz file you might run it through "zcat backup.sql.gz | dbsake sieve [options...]"

New in version 2.0.0.

-z, --compress-command <command>
Filter output files through this command. If `--format` is not set to 'directory', then this option is ignored. The sieve command will detect most common compression command and create an appropriate extension on the output files. For example, –compress-command=gzip will create .sql.gz files under the path specified by `--directory`.

Defaults to "gzip -1".

Changed in version 2.0.0: -f/–filter-command was renamed to -z/–compress-command

-t, --table <glob pattern>
f `--table` is specified, then only tables matching the provided glob pattern will be included in the output of the sieve command. Each table is qualified by the database name in "database.table" format and then compared against the glob pattern. For example, to include all tables in the "mysql" database you would specify –table="mysql.*".

This option may be specified multiple times and sieve will include any table that matches at least one of these options so long as the table does not also match an `--exclude-table` option.

If no –table options are provided, all tables are included in the output that do not otherwise match an `--exclude-table` pattern.

New in version 2.0.0.

-T, --exclude-table <glob pattern>
If --exclude-table is specified, then only tables not matching the provided glob pattern will be included in the output of the sieve command. Each table is qualified by the database name in "database.table" format and then compared against the glob pattern. For example, to exclude the mysql.user table from output you would specify the option: "–exclude-table=mysql.user".

This option may be specified multiple times and sieve will include any table that matches at least one of these options so long as the table does not also match an --exclude-table option.

If no --exclude-table options are provided, all tables are included in the output that match at least one --table pattern, or all output is included if neither --exclude-table or --table options are provided.

New in version 2.0.0.

--defer-indexes
This option rewrites the output of CREATE TABLE statements and arranges for secondary indexes to be created after the table data is loaded. This causes an additional ALTER TABLE statement to be output after the table data section of each table, when there is at least one secondary index to be added.

If there are foreign key constraints on the table, associated indexes will not be deferred unless the --defer-foreign-keys option is also specified.

This option only applies to InnoDB tables and is only efficient on MySQL 5.1+ (if the innodb plugin is enabled) or on MySQL 5.5+ (default InnoDB engine), where the fast alter path may be used.

--defer-foreign-keys
This option rewrites the output of CREATE TABLE statements and adds foreign key constraints after the table data is loaded. This is primarily useful to allow deferring secondary indexes with associated foreign keys.

This option only makes sense if reloading a dump into MySQL 5.6+, othrewise adding indexes will require a full table rebuild and will end up being much slower than just reloading the mysqldump unaltered.

--write-binlog / --no-write-binlog
If --no-write-binlog is set, sieve will output a SET SQL_LOG_BIN=0 SQL command to the beginning of the dump to avoid writing to the binary log when reloading the resulting output. Use the option with care, as the resulting dump will not replicate to a slave if this option is set.

New in version 2.0.0.

--table-schema / --no-table-schema
If --no-table-schema is used, sieve will not output any CREATE TABLE statements and will not output any CREATE VIEW statements. Only table data, routines and events will be output (as dictated by other options).

New in version 2.0.0.

--table-data / --no-table-data
If --skip-table-data is set, sieve will not output any table data sections and only output DDL. Reloading such a dump will result in empty tables.

New in version 2.0.0.

--master-data / --no-master-data
If the --master-data option is set, any commented out CHANGE MASTER statements will be uncommented in the output. This is useful of setting up a replication slave from a backup created using –master-data=2.

If the --no-master-data option is set, any CHANGE MASTER statements will be commented out in the output, ensuring no CHANGE MASTER is run. This is useful for dumps created with –master-data[=1].

New in version 2.0.0.

--routines / --no-routines
Include or exclude routines from the output, if routines were found in the input file. By default routines are not

excluded and will ony be excluded if the –no-routines option is specified. The –routines option used to cancel a previous –no-routines option.

New in version 2.0.0.

--events / --no-events
Include or exclude events from the output, if events were found in the input file. By default events are not excluded and will ony be excluded if the –no-events option is specified. The –events option can be used to cancel a previous –no-events option.

New in version 2.0.0.

--triggers / --no-triggers
Include or exclude table triggers from the output, if triggers were found in the input file. By default triggers are included for any output tables (subject to table filtering). –no-triggers will disable output for all triggers and –triggers can be used to cancel the effects of an earlier –no-triggers option.

New in version 2.0.0.

-O, --to-stdout
The --to-stdout option will force output to be written to stdout even if stdout appears to be an active terminal. This can be useful in cases when filtering the mysqldump output or when not outputing large amounts of data and want to read it directly on the terminal. By default, the sieve command will abort if it detects that it would output to a terminal and –to-stdout is not used.

3.6 upgrade-mycnf

Copy a my.cnf file and patch any deprecated options.

This command is used to rewrite a my.cnf file and either strip out or rewrite options that are not compatible with a newer version of MySQL.

The original my.cnf is left untouched. A new my.cnf is output on stdout and reasons for rewriting or excluding options are output on stderr.

If -p, –patch is specified a unified diff is output on stdout rather than a full my.cnf. –patch is required if a my.cnf includes any !include* directives.

3.6.1 Usage

```
Usage: dbsake upgrade-mycnf [OPTIONS]

  Upgrade a MySQL option file

Options:
  -c, --config PATH               my.cnf file to parse
  -t, --target [5.1|5.5|5.6|5.7]  MySQL version to target
  -p, --patch                     Output unified diff rather than full config
  -?, --help                      Show this message and exit.
```

3.6.2 Example

```
$ dbsake upgrade-mycnf -t 5.6 --patch -c /etc/my.cnf
2014-01-04 05:36:34,757 Removing option 'skip-external-locking'. Reason: Default␣
↪behavior in MySQL 4.1+
--- a/etc/my.cnf
+++ b/etc/my.cnf
@@ -17,7 +17,6 @@
 datadir                         = /var/lib/mysql
 #tmpdir                         = /var/lib/mysqltmp
 socket                          = /var/lib/mysql/mysql.sock
-skip-external-locking           = 1
 open-files-limit                = 20000
 #sql-mode                       = TRADITIONAL
 #event-scheduler                = 1
```

3.6.3 Options

-c <config>, **--config** <config>
: Specify which my.cnf file to process Defaults to /etc/my.cnf

-t <version>, **--target** <version>
: Specify which version of MySQL to target. This controls which options are rewritten based on the deprecated options in the target MySQL version. Defaults to 5.5

-p, --patch
: Specify the output should be a unified diff rather than a full my.cnf. Defaults to outputting a full my.cnf if this option is not specified.

3.7 fincore

Discover which parts of a file are cached by the OS.

This command uses the mincore() system call on linux to grab a mapping of cached pages. Currently this done with a single mincore() call and requires 1-byte for each 4KiB page. For very large files, this may require several MiBs or more of memory. For a 1TB file this is 256MiB, for instance.

3.7.1 Usage

```
Usage: dbsake fincore [OPTIONS] [PATHS]...

  Report cached pages for a file.

Options:
  -v, --verbose
  -?, --help     Show this message and exit.
```

3.7.2 Example

```
# dbsake fincore /var/lib/mysql/ibdata1
/var/lib/mysql/ibdata1: total_pages=6656 cached=0 percent=0.00
# cat /var/lib/mysql/ibdata1 > /dev/null
```

(continues on next page)

(continued from previous page)

```
# dbsake fincore /var/lib/mysql/ibdata1
/var/lib/mysql/ibdata1: total_pages=6656 cached=6656 percent=100.00
```

3.7.3 Options

--verbose
Print each cached page number that is cached.

path [path...]
Path(s) to check for cached pages

3.8 uncache

Remove a file's contents from the OS cache.

This command is useful when using O_DIRECT. A file cached by the OS often causes O_DIRECT to use a slower path - and often buffered + direct I/O is an unsafe operation anyway.

With MySQL, for instance, a file may be accidentally cached by filesystem backups that just archive all files under the MySQL datadir. MySQL itself may be using innodb-flush-method=O_DIRECT, and once these pages are cached there can be a performance degradation. uncache drops these cached pages from the OS so O_DIRECT can work better.

3.8.1 Usage

```
Usage: dbsake uncache [OPTIONS] [PATHS]...

  Drop OS cached pages for a file.

Options:
  -?, --help  Show this message and exit.
```

3.8.2 Example

```
# dbsake fincore /var/lib/mysql/ibdata1
/var/lib/mysql/ibdata1: total_pages=6656 cached=6656 percent=100.00
# dbsake uncache /var/lib/mysql/ibdata1
Uncached /var/lib/mysql/ibdata1
# dbsake fincore /var/lib/mysql/ibdata1
/var/lib/mysql/ibdata1: total_pages=6656 cached=0 percent=0.00
```

3.8.3 Options

path [path...]
Path(s) to remove from cache.

3.9 unpack

New in version 2.1.0.

Unpack a datadir archive for MySQL.

This expects a .tar or .xb based file archive of a MySQL datadir, whose paths are relative to the MySQL datadir. Such archives might be generated by Percona XtraBackup or other LVM based backup utilities.

3.9.1 Usage

```
Usage: dbsake unpack [options] [path]

  Unpack a MySQL backup archive.

  This command will unpack tar or Percona XtraBackup xbstream archives with
  support for filtering and extracting only a subset of tables.

Options:
  -l, --list-contents             List the contents of the archive, but don't
                                  extract.
  --progress / --no-progress      Enable/disable progress bar when unpacking.
  -C, --directory <path>          Directory to output to (default: $PWD)
  -t, --table <db.table>          Only extract table datafiles matching
                                  specified database.table glob patterns.
  -T, --exclude-table <db.table>  Exclude table data files matching specified
                                  databsae.table glob patterns.
  -?, --help                      Show this message and exit.
```

3.9.2 Example

```
$ dbsake unpack -C /data/mysql/ < backup.xb.gz
...
```

3.9.3 Options

-l, --list-contents
List the contents of the archive but do not extract any file contents. Each file path will be output to stdout. Table inclusion/exclusion options are honored and any excluded table will not output.

--progress / --no-progress
Enable or disable (respectively) progress bar output. This outputs a bar on stderr indicating how much data has been read thus far, and, if known, an estimated ETA until completion.

If stderr appears to be a TTY (i.e. isatty(3) is true for stderr), progress wil be enabled by default.

-C, --directory <path>
Output all archived files relative to <path>.

<path> defaults to the current working directory.

-t, --table <glob>
Restricted extracted table data files to those who match a database.tablename glob. This matching is done

against the decoded tablename so paths like foo@002dbar/foo@002dbaz would be filtered with a pattern like 'foo-bar.foo-baz'.

This option may be specified multiple times. A table is included if it matches at least one include pattern and does not match any exclude patterns.

Note: mysql.* is always enabled regardless of this option. To exclude the mysql schema, a specified –exclude-table option should be used.

-T, --exclude-table <glob>
Restricted extracted table data files to those who do NOT match a database.tablename glob. This matching is done against the decoded tablename after processing the MySQL filename encoding and after removing any relative extensions or partitioning information from the filename.

This option may be specified multiple times. A table is extracted if it matches at least one include option (if any are specified) and does not match any exclude options.

path
Path to the archive to process. This defaults to stdin but the unpack command will refuse to process input from a tty. You must redirect stdin with a valid archive file or specify a path to a valid archive.

Unpack supports both xbstream format files (as generated by Percona XtraBackup) and tar format files. All paths are assumed to be relative to the datadir (similar to archives generated by Percona XtraBackup or various Holland Backup Manager plugins). dbsake will transparently decompress input archives - currently gzip, bzip2, lzop and xz extension are supported, provided the decompression utilities are available on $PATH.

CHAPTER 4

History

4.1 2.2.0 (2019-01-29)

Bugs fixed

- sandbox now handles MySQL 5.7 instance bootstrapping (PR #128)
- sieve will now use pigz if available when –format=directory is specified (issue #130)
- python2.6 is no longer tested

4.2 2.1.2 (2017-02-15)

Bugs fixed:

- dbsake 2.1.1 sandbox command was broken due to incorrectly quoting the –ledir argument when generating sandbox.sh (issue #120)

4.3 2.1.1 (2017-02-07)

New features

- Add frmdump –recursive option to dump all *.frm files found in the target directory (issue #60)

Bugs fixed:

- frmdump now handles tokudb row_format from .frm files created by Percona Server (issue #80)
- sandbox generated sandbox.sh script would previously wait for the mysqld socket to appear as part of the “start” action, which was problematic for Galera based binaries. sandbox.sh start now waits for the pid-file to appear via the “sandbox.sh status” action. (issue #88)

- unpack command had a bug that failed on uncompressed input; similarly affected the sandbox command when unpacking uncompressed archives via the –data-source / -s option (issue #90)
- upgrade-mycnf now handles paths with spaces (issue #94)
- python-setuptools is now required for install (issue #95)
- Remove unicode_literals usage in cli to avoid noisy warnings from click (Issue #96)
- contrib/ is now included in the source package (PR#98 from azlev)
- frmdump failed to decode tables specified with utf8mb4, binary or other less common encodings. (issues #97, #102)
- starting a sandbox failed on recent versions of MySQL due to the mysqld_safe ledir option now being prohibited in config files. (issue #103)

4.4 2.1.0 (2015-01-28)

New features

- unpack command added to help extracting files from .tar or .xb archives See http://docs.dbsake.net/en/latest/commands/unpack.html for for details.
- "make dbsake.sh" would fail under python2.6 due to some assumptions around python's zipfile module.
- "make test-all" will now test against python2.6, python3.4 environments, if available.

Bugs fixed:

- fincore: handle io errors more gracefully (issue #72)
- **frmdump: decoding/encoding _filename encoding MySQL names is now** compatible with python3. This would previously fail under python3 in some circumstances.
- **sandbox: sandbox.sh no longer uses the sed -r flag when processing my.cnf** options to make the script more compatible on non-GNU platforms. (issue #70)
- **sandbox: gpg stderr output was previously logged incorrectly** (issue #74)
- **sandbox: mysqld –init-file is now used to generate the database user** rather than parsing the user.frm or injecting SQL into the bootstrap process. This resolves an issue with recent MySQL 5.7 releases and should generally be more robust.
- **sandbox: "sandbox.sh mysql" now sets MYSQL_HISTFILE to .mysql_history** relative to the sandbox base directory, rather than appending to ~/.mysql_history. This avoids problems mixing libedit w/ libreadline (issue #76)
- **sandbox: stale pidfiles are now detected and handled gracefully.** Previously a stale pidfile would require manual intervention to remove the mysql.pid or ./sandbox.sh start would fail to automatically restart a crashed instance. (issue #75)
- **sandbox: The –datasource option now correctly display progress** when unpacking a compressed datasource. (issue #73)
- **sandbox: The –datasource option now handles xbstream archives in** addition to .tar archives and supports more compression options for both (.gz, .bz2, .xz and .lzo)
- **sandbox: –progress / –no-progress options have been added to** control the display of progressbars
- **sandbox: logging more consistently uses whitespace when subtasks** are completed during sandbox creation.

- **sandbox: The generated my.sandbox.cnf now generates a somewhat** cleanear default config. wait-timeout / interactive-timeout now use the MySQL defaults rather than being 600 / 3600 (respectively). The buggy relay-log-space-limit is avoided and innodb-buffer-pool-{dump,restore} options are set by default on MySQL 5.6+.
- **sandbox: Extracting –datasource archives are now handled via the** internal unpack command for consistency.
- **sieve: Decompressing compressed input would fail on platforms where** flushing read-only files results in an EBADF file. (Issue #71)
- **sieve: documentation incorrectly referenced "–no-write-binlog" as** "–disable-binlog" (issue #81)
- **sieve: mariadb gtid information in mysqldump output is now handled** properly (issue #78)
- **upgrade-mycnf: The example in the documentation was incorrectly missing** the -c / –config option. (issue #82)

4.5 2.0.0 (2014-08-05)

The 2.0.0 release is a major update to dbsake significantly updating various internals and introducing some backwards incompatible changes.

As of 2.0.0, dbsake uses semantic versioning and new features will only be introduced in point releases (2.1, 2.2, 2.3, etc.) Only strict bug fixes will be introduced in patch releases (2.0.1, 2.0.2, etc.) going forward. Incompatible changes will only be introduced in major version bumps (3.0, 4.0, etc.).

Compatibility changes:

- frm-to-schema command has been renamed to frmdump
- frmdump -r/–raw-types option was renamed to -t/–type-codes
- mysql-sandbox command has been renamed to sandbox
- filename-to-tablename command has been renamed to decode-tablename
- tablename-to-filename command has been renamed to encode-tablename
- importfrm command has been removed
- read-ibbinlog command has been removed
- split-mysqldump has been completely redesigned and renamed to "sieve", with many more capabilities than the old split-mysqldump command. Read the sieve documentation for more information.
- dbsake 2.0+ uses click for option parsing instead of baker.py used in 1.0. This provides a more standard option parsing experience, but this means dbake no longer accepts position arguments interchangably with options.
- The sandbox command now uses jinja2 to generate templates rather than tempita.
- sandbox -D is now a short option for –datadir. Use -s as a short option for –data-source.
- sandbox –prompt-password was shortened to simply –password
- dbsake no longer uses the sarge library internally
- dbsake no longer uses the tempita library internally

New features:

- dbsake now supports bash completion via click. See Enable bash completion for details.

- sandbox now uses system compression commands to decompress tarballs from the –data-source option rather than strictly relying on the python standard library. This should speed up creating a sandbox from existing data in some cases and supports more compression formats (.gz,.bz2, .lzo, .xz) (Issue #64)
- sandbox now includes the mysql.* schema by default when performing partial restores from existing data (e.g. -D backup.tar.gz -t mydb.*). Restoring mysql tables to the sandbox can be suppressed with the -T / –exclude-table 'mysql.*' option. (Issue #67)
- sandbox now generates a simplified sandbox.sh shell script file. The sandbox.sh script now read mysql server options from the my.sandbox.cnf config file rather than hardcoding various options in sandbox.sh. This would previously make it tedious to change the path for log-error or other options.
- sandbox no longer generates a sandbox.sh which sources /etc/sysconfig.
- sandbox now supports a -u/–mysql-user option for specifying the database user created during sandbox setup.
- sandbox now supports a -D / –datadir option for specifying the MySQL datadir that should be used for a sandbox. This supersedes support for –data-source=<directory>, which now only supports tarball targets.
- frmdump now handles MariaDB microsecond precision date/time types.
- fincore and uncache no longer fail when no paths are passed. This usage is now considered a no-op.

Bugs fixed:

- sandbox failed to create ./tmp/ when overwriting an existing sandbox directory with –force, if ./data/ already existed but ./tmp did not. (Issue #65)
- sandbox now handles 5.0 / 5.1 binary tarball installs more robustly. Previously, mysqld_safe would fail to find my_print_defaults in the sandbox directory and could fail if sandbox.sh was run when the current working directory != sandbox directory. (Issue #66)
- frmdump incorrectly defaulted to SQL SECURITY INVOKER when decoding view .frm files. This behavior has been changed to use MySQL's default of SQL SECURITY DEFINER.
- frmdump did not match MySQL output when decoding views
- frmdump did not correctly decode default values for 3-byte MEDIUM int fields due to several logic errors.
- frmdump did not include the unsigned attribute for float / double fields which were defined with a (precision, scale) scale attribute.
- frmdump did not format MariaDB TIME fields with microsecond precision correctly.
- frmdump did not format MariaDB TIMESTAMP fields with microsecond precision correctly.
- frmdump did not format MariaDB DATETIME(N) with microsecond precision correctly.
- frmdump did not handle timestamp values that defaulted to '0' correctly, and instead used '1970-01-01 00:00:00' as the default, rather than the MySQL convention of using '0000-00-00 00:00:00'
- frmdump did not always format microseconds for MySQL 5.6 DATETIME(N) fields correctly.

4.6 1.0.9 (2014-07-09)

New features:

- mysql-sandbox now provides a –force option to disable various sanity checks allowing installing into an existing directory (issue #47)
- mysql-sandbox now provides a –prompt-password option for setting the root@localhost password for a new sandbox. This is a boolean option that will either prompt for a password (if stdin is attached to a TTY) or read the password directly from stdin. (issue #53)

- mysql-sandbox now generates my.sandbox.cnf with relay-log and bin-log options relative to the datadir. These options are still commented out by default, but now do not reference the non-standard /var/lib/mysqllogs path. (issue #51)
- mysql-sandbox now includes a commented out "#port = <version>" option in the generated my.sandbox.cnf options file. (issue #55)
- mysql-sandbo now provides a –innobackupex-options/-x option to allow passing arbitrary options to innobackupex –apply-log when bootstrapping a sandbox from an xtrabackup tarball backup image (issue #56)

Bugs fixed:

- mysql-sandbox now includes a comment indiciating the version of dbsake in both the generated sandbox.sh and my.sandbox.cnf files (issue #42)
- mysql-sandbox now reports errors better when a binary tarball cannot be found on the MySQL CDN (issue #44)
- mysql-sandbox now provides more details when encountering a bad mysql tarball distribution (issue #46)
- mysql-sandbox no longer raises an unchecked exception when –data-source specifies a datadir without an ib_logfile (issue #49)
- mysql-sandbox now bootstraps sandboxes with default-storage-engine=MyISAM in order to handle TokuDB binary tarball distributions better (issue #50)
- mysql-sandbox now sets the no-auto-rehash option for the mysql client in my.sandboc.cnf's [mysql] section.
- mysql-sandbox now only sets the mysql.user plugin field to 'mysql_native_password' for MySQL 5.7. This otherwise causes issues for MariaDB when bootstrapping MariaDB from MySQL 5.6+ data. (issue #54)
- frm-to-schema no longer fails when using the –raw-types option. This was broken in v1.0.8 as part of a fix for issue #38. (issue #45)

4.7 1.0.8 (2014-04-02)

Bug fixes:

- mysql-sandbox now fails more gracefully if bootstrap files are invalid or not found in a MySQL distribution (issue #37)
- mysql-sandbox now correctly uses /usr/share/percona-server rather than trying to use a missing or incorrect /usr/share/mysql for system installs of Percona Server (issue #41)
- mysql-sandbox is now less chatty and many less critical details are only logged with dbsake –debug to reduce spam
- frm-to-schema now correctly decodes default values for old MySQL varchar columns generated by servers prior to MySQL 5.0. (issue #36)
- frm-to-schema now decodes unicode metadata identifiers correctly rather than failing on a parsing error (issue #38)
- frm-to-schema now formats TEXT types (tinytext, mediumtext, text, longtext) with the associated column level charset or collation (issue #40)
- split-mysqldump nows correctly handles dump files generated with mysqldump –flush-privileges (issue #33)
- split-mysqldump now handles a commented CHANGE MASTER line generated by mysqldump –master-data=2 (issue #33)

4.8 1.0.7 (2014-02-20)

Bug fixes:

- dbsake frm-to-schema now reads signed MEDIUMINT default values; Previously a bug caused an uncaught exception to be thrown (issue #19)
- dbsake frm-to-schema now interprets negative signed MEDIUMINT default values correctly; Previously this would result in incorrect values (issue #23)
- dbsake frm-to-schema introduced a bug in v1.0.6 that caused an exception when formatting BIGINT default values (issue #20)
- dbsake frm-to-schema should now handle nullable columns more robustly; This addresses the improper fix made in v1.0.6 for issue #9. Previously this command was not honoring all the table handler options resulting in spuriously misinterpretting a column's default value as NULL. (issue #21)
- dbsake frm-to-schema has improved the formatting for float/double column's default values; Previously this used default python precision in output which was often inaccurate for 'float' and generally did not match the output from mysql SHOW CREATE TABLE (issue #22)
- dbsake frm-to-schema now display table comments similar to SHOW CREATE TABLE Previously this was displayed with a space separator as "COMMENT '<value>'" but now is display as "COMMENT='<value>'" (issue #24)
- dbsake frm-to-schema now displays decimal default values correctly in cases where the encoded decimal bytes were not a multiple of 4 (issue #26)
- dbsake frm-to-schema now trims insignificant zeros from the interger part of a decimal value; Previously this would display decimal(19, 0) default '0' as default '000' due to implementation details of the decoding algorithm (issue #27)
- dbsake mysql-sandbox now checks for the existence of mysql installation .sql scripts; Previously this woudl result in an uncaught exception if /usr/share/mysql existed but the files necessary for bootstrapping did not (issue #25)
- dbsake mysql-sandbox now creates the performance_schema database and tables under MariaDB 5.5+ (issue #28)

4.9 1.0.6 (2014-02-17)

New features:

- dbsake mysql-sandbox's generated ./sandbox.sh start/stop actions now show progress more visibly by echoing a '.' once a second until the start/stop action finishes (issue #18)

Bugs fixed:

- dbsake now parses boolean options correctly; previously these would sometimes consume the next argument in the commandline (issue #8)
- dbsake split-mysqldump now supports deferring indexes specified with an algorithm; previously these weren't matched correctly and thus would never be deferred.
- dbsake split-mysqldump now aborts if an invalid mysqldump header is detected. previously it was queing lines looking for the end of the header and used excessive memory and ultimately failing (issue #17)
- dbsake frm-to-schema now handles null values for blob types (issue #9)

- dbsake frm-to-schema now quotes integer default values; Previously a default of 0 was unquoted and would be handled identically to a missing default value (issue #11)
- dbsake frm-to-schema now handles MySQL 5.0 .frm files; Previously frm-to-schema would attempt to read a non-existent partitioning clause and fail. (issue #14)
- dbsake mysql-sandbox now auto-detects innodb-data-file-path based on existing ibdata* files from –data-source, or uses MySQL default if this is an empty sandbox instance (issue #12)
- dbsake mysql-sandbox now handles invalid mysqld binaries more gracefully; This may occur if attempting to run i686 on an x86_64 platform for instance. Previously this would fail on an ENOENT error and an uncaught exception would be thrown. (issue #13)
- dbsake mysql-sandbox –sandbox-directory now handles relatives paths; Previously these were passed as-is to mysql which would reevaluate the path relative to the sandbox directory and typically fail to start (issue #15)

4.10 1.0.5 (2014-01-31)

New features:

- dbsake mysql-sandbox's generated ./sandbox.sh script now supports an 'upgrade' action to run mysql_upgrade against the sandbox instance. (issue #1)
- dbsake mysql-sandbox –mysql-distribution=system (the default) now only copies the mysqld binary and assumes all other utilities are in the path; mysqld is copied to avoid security issues under apparmor in debuntu environments
- dbsake mysql-sandbox has reduced the required disk footprint of mysql distribution tarballs by excluding ./bin/*_embedded and ./bin/mysql-debug binaries in addition to excluding ./mysql-test, ./include and ./sql-bench that was done previously.
- dbsake mysql-sandbox –data-source now supports directory paths, which point to an existing MySQL datadir; This option simply symlinks the specified directory to the sandbox ./data path. Sandbox creation will fail if any of the standard InnoDB data/log files are locked indicating they are already used by another active instance.
- dbsake mysql-sandbox will now set the root@localhost plugin to 'mysql_native_password' when setting a password. This avoids an issue with MySQL 5.7 which refuses authentication if plugin is not set, which may be the case if a sandbox is loaded with data from an earlier version.
- dbsake mysql-sandbox now checks for libaio as part of the setup process and will abort if this is not available for MySQL 5.5+; This check can be disabled with the –skip-libcheck option, but if mysqld requires this library the sandbox creation will still fail in this case.
- dbsake mysql-sandbox now performs gpg verification against downloaded mysql distribution tarballs using mysql.com's public key; This behavior can be disabled by using the new –skip-gpgcheck option
- dbsake mysql-sandbox's generated ./sandbox.sh script now supports a 'metadata' action for dumping information about the sandbox environment
- dbsake mysql-sandbox's generated ./sandbox.sh script now supports a 'version' action to echo the mysql version the sandbox was installed with

Bugs fixed:

- dbsake mysql-sandbox no longer suppresses stderr when running mysqld –version; This is done to discover the exact version of the deployed mysql distribution to allow my.cnf generation to make adjustments based on the features available.
- dbsake mysql-sandbox's generated ./sandbox.sh script now accepts extra commandline options for the 'restart' action which behaves identically to the 'start' action - these are passed down to the mysqld_safe script

4.11 1.0.4 (2014-01-24)

New features:

- dbsake now handles SIGINT gracefully
- dbsake now logs a cleaner format
- dbsake –log-level option removed; –debug / –quiet options were added as simpler knobs to tweak logging output
- dbsake now longer depends on argparse and it has been removed from the source tree
- dbsake mysql-sandbox has renamed the –mysql-source option to –mysql-distribution; the short option (-m) is unchanged
- dbsake mysql-sandbox –data-sourcel-D <path> option added with support for LVM and xtrabackup tarballs
- dbsake mysql-sandbox –tablel-t / –exclude-tablel-T <pattern> option added to filter files read from –data-source tarballs
- dbsake mysql-sandbox –cache-policy option added to support caching downloaded MySQL distribution tarballs
- dbsake mysql-sandbox now supports a progress bar when downloading mysql tarball distributions and when extracting –data-source tarballs; The progress bar is only displayed when stderr is attached to a tty
- dbsake mysql-sandbox now emits timing information for each major step in the sandbox creation process
- dbsake mysql-sandbox's generated ./sandbox.sh script now supports 'use' and 'mysql' actions for connecting to the sandbox instance; These are aliases for the 'shell' command included in v1.0.3
- dbsake mysql-sandbox's generated ./sandbox.sh script now supports a 'mysqldump' action for trivially running mysqldump against the sandbox instance
- dbsake mysql-sandbox's generated ./sandbox.sh script now supports arguments for the 'start' action - these are passed directly to the mysqld_safe process to enable additional mysql options on startup
- dbsake mysql-sandbox's generated ./sandbox.sh script now supports an 'install-service' action that will deploy the ./sandbox.sh as a standard SysV initscript

Bugs fixed:

- dbsake mysql-sandbox no longer prunes users in the sandbox to avoid removing existing users from user-provided –data-source tarballs

4.12 1.0.3 (2014-01-16)

New features:

- third-party sarge[1] package added to dbsake tree
- third-party tempita[2] package added to dbsake tree
- dbsake now "lazy loads" imports for most commands to improve initial startup times
- dbsake mysql-sandbox command added; see documentation for more details

Bugs fixed:

- dbsake frm-to-schema now supports very old VARCHAR fields (MYSQL_TYPE_VAR_STRING)
- dbsake.spec now supports building under EPEL 5 environments

[1] https://pypi.python.org/pypi/sarge/0.1.3
[2] https://pypi.python.org/pypi/Tempita/0.5.3dev

4.13 1.0.2 (2014-01-07)

New features:

- dbsake frm-to-schema now parses views from plaintext .frm files
- dbsake frm-to-schema –replace option added; This outputs view definitions as CREATE OR REPLACE view to ease importing into MySQL
- dbsake frm-to-schema –raw-types option added; This adds comments to the column output indicating the low-level raw mysql type (e.g. MYSQL_TYPE_TINYBLOB) - previously these were always displayed
- dbsake frm-to-schema now outputs a mysqldump-like comment block before each table or view's DDL

Bugs fixed:

- dbsake frm-to-schema now formats prefix indexes correctly
- dbsake frm-to-schema no longer outputs MYSQL_TYPE_* comments in CREATE TABLE output by default; use the new –raw-types to see this information.

4.14 1.0.1 (2014-01-06)

New features: rename CHANGES.rst -> HISTORY.rst

- dbsake –version/-V option added
- documentation has been added to the project

Bugs fixed:

- dbsake –log-level now recognizes log level names correctly
- dbsake fincore now handles zero-byte files gracefully
- dbsake fincore now releases mmap resources gracefully
- dbsake {fincore,uncache} now skip paths that are not a regular file
- dbsake.spec RPM spec now properly depends on python-setuptools

4.15 1.0.0 (2014-01-02)

- First release of dbsake

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/abg/dbsake/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" is open to whoever wants to implement it.

5.1.4 Write Documentation

dbsake could always use more documentation, whether as part of the official dbsake docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/abg/dbsake/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *dbsake* for local development.

1. Fork the *dbsake* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/dbsake.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv dbsake
$ cd dbsake/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 dbsake tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, 3.4, and for PyPy. Check https://travis-ci.org/abg/dbsake/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests/test_dbsake.py
```

CHAPTER 6

Appendix

6.1 Description of the .frm format

dbsake parses some undocumented / poorly documented formats for MySQL which were decoded by inspecting the MySQL source code. Here is a detailed document that attempts to describe the details of such formats for other who may hack on these parts of dbsake.

6.1.1 .frm fileinfo section

The fileinfo section consists of 64 bytes that encode information about the rest of the .frm file and various table level options.

The table here describes the information each byte encodes. Offsets are from the beginning of the file. All values are little-endian integer of the size noted in the Length column.

Offset		Length (bytes)	Description
Hex	Dec		
0000	0	2 bytes	"Magic" identifier Always the byte sequence fe 01
0001	1		
0002	2	1 byte	.frm version[1]
0003	3	1 byte	"legacy_db_type"[2]
0004	4	2-bytes	"names_length" - always 3 and not used in recent MySQL. MySQL 3.23 set this to 1
0005	5		
0006	6	2-bytes	IO_SIZE; Always 4096 (0010)
0007	7		
0008	8	2-bytes	number of "forms" in the .frm Should always be 1, even back to 3.23
0009	9		

Continued on next page

Table 1 – continued from previous page

Offset		Length (bytes)	Description
Hex	Dec		
000a	10	4-bytes	Not really used except in .frm creation Purpose unclear, i guess for aligning sections in the ancient unireg format
000b	11		
000c	12		
000d	13		
000e	14	2-bytes	"tmp_key_length"; if equal to 0xffff then the key length is a 4-byte integer at offset 0x002f
000f	15		
0010	16	2-bytes	"rec_length" - this is the size of the byte string where default values are stored See Default Values
0011	17		
0012	18	4-bytes	Table MAX_ROWS=N opton
0013	19		
0014	20		
0015	21		
0016	22	4-bytes	Table MIN_ROWS=N option
0017	23		
0018	24		
0019	25		
001a	26	1-byte	Unused - always zero in 3.23 through 5.6
001b	27	1-byte	Always 2 - "// Use long pack-fields"
001c	28	2-bytes	key_info_length - size in bytes of the keyinfo section
001d	29		
001e	30	2-bytes	create_info->table_options See HA_OPTION_* values in include/my_base.h
001f	31		
0020	32	1-byte	Unused; comment "// No filename anymore"
0021	33	1-byte	5 in 5.0+ comment "// Mark for 5.0 frm file"
0022	34	4-bytes	Table AVG_ROW_LENGTH option
0023	35		
0024	36		
0025	37		
0026	38	1-byte	Table DEFAULT CHARACTER SET option[3]
0027	39	1-byte	Unused[4]
0028	40	1-byte	Table ROW_FORMAT option
0029	41	1-byte	Unused; formerly Table RAID_TYPE option
002a	42	1-byte	Unused; formerly Table RAID_CHUNKS option
002b	43	4-bytes	Unused; formerly Table RAID_CHUNKSIZE option

Continued on next page

Table 1 – continued from previous page

Offset		Length (bytes)	Description
Hex	Dec		
002c	44		
002d	45		
002e	46		
002f	47	4-bytes	Size in bytes of the key-info section where index metadata is defined
0030	48		
0031	49		
0032	50		
0033	51	4-bytes	MySQL version encoded as a 4-byte integer in little endian format. This is the value MYSQL_VERSION_ID from include/mysql_version.h in the mysql source tree. Example: 'xb6xc5x00x00' 0x0000c5b6 => 50614 => MySQL v5.6.14
0034	52		
0035	53		
0036	54		
0037	55	4-bytes	Size in bytes of table "extra info" • CONNECTION=<string> (FEDERATED tables) • ENGINE=<string> • PARTITION BY clause + partitioning flags • WITH PARSER names (MySQL 5.1+) • Table COMMENT[5]
0038	56		
0039	57		
003a	58		
003b	59	2-byte	extra_rec_buf_length
003c	60		
003d	61	1-byte	Storage engine if table is partitioned[6]
003e	62	2-bytes	Table KEY_BLOCK_SIZE option
003f	63		

[1] This is defined as FRM_VER+3+ test(create_info->varchar) in 5.0+ Where FRM_VER is defined as 6, so the frm version will be either 9 or 10 depending on if the table has varchar columns

[2] Maps to an enum value from "enum legacy_db_type" in sql/handler.h

[3] Character set id maps to an id from INFORMATION_SCHEMA.COLLATIONS and encodes both the character set name and the collation

[4] In the source code, there is a comment indicating this byte will be used for TRANSACTIONAL and PAGE_CHECKSUM table options in the future

[5] The table comment is stored in one of two places in the .frm file If the comment size in bytes is < 255 this is stored in the forminfo Otherwise it will be estored in the extra info section after the fulltext parser names (if any)

[6] Numeric id that maps to a enum value from "enum legacy_db_type" in sql/handler.h, similar to legacy_db_type

6.1.2 Key info section

The key info section should always start at offset 0x1000 (4096); this is obtained from the 2-byte integer in fileinfo header at offset 6, but in any version of MySQL in the past decade will be 4096.

The size in bytes of this section is obtained from the key_length - typically this is 4-byte integer at offset 0x002f (47) in the header. Older versions of MySQL only allocated a 2-byte integer for this length, at offset 0x000e (14). This old location will have the value 0xffff if the key info length exceeds the capacity of a 2-byte integer.

The structure of this section consists of an initial header noting the total number of keys, total number of key components and the size of "extra" key data (namely index names and index comments). This is followed by a group for each index defined in the table and then the extra data - names for each index followed by an optional index comment strings.

The header is essentially three integers:

[key_count][key_parts_count][length of extra data]

Where key_count is the number of indexes this metdata describes, key_parts_count is the number of components across all indexes and the length of extra data indicates how many bytes the index names and comments uses.

key_count and key_parts_count may be either 1 or 2 bytes. If the first byte is > 128 then key_count and key_parts_count use two bytes, otherwise they use one byte each. The extra length is always a 2 byte integer.

The logic in dbsake is:

```
key_count = keyinfo.uint8()
if key_count < 128:
    key_parts_count = keyinfo.uint8()
    keyinfo.skip(2)
else:
    key_count = (key_count & 0x7f) | (keyinfo.uint8() << 7)
    key_parts_count = keyinfo.uint16()
key_extra_length = keyinfo.uint16()
```

Each key metadata consists of 8 bytes and each key part consists of 9 bytes. So the total length of the index metadata is calculated by the formula:

```
key_count * 8 + key_parts_count * 9
```

And this is the offset, relative to the start of keyinfo section, where the index names and comments are found.

Each index group consists of 8 bytes of key metadata followed by 9 bytes of metadata for each indexed column.

Index metadata (8 bytes)		
flags	2 bytes	key flags from include/my_base.h.
length	2 bytes	length of the index
key parts	1 byte	number of columns covered by this index
algorithm	1 byte	Key algorithm - maps to enum value "enum ha_key_alg"
unused	2 bytes	

Followed by 1 or more column index metadata:

Column index metadata (9 bytes)		
field number	2 bytes	Which column is indexed
offset	2 bytes	Offset into a MySQL datastructure (internal use)
unused	1 byte	
key_type	2 bytes	maps to enum ha_base_keytype
length	2 bytes	length of this index component

The names and comments follow this data with names being separated by the byte value 255 ('\xff') and the names and comments sections being separated by a null byte. So this essentially looks like this sort of python bytestring:

```
b'\xffPRIMARY\xffix_column1\xff\x00<index comments>'
```

Index comments are length-prefixed strings. So there is a 2 byte integer (little-endian) followed by the specified number of bytes for each comment.

Index comments are not terribly common so this will often be empty.

6.1.3 Defaults Section

Immediately after the keyinfo section there is a byte string that details the defaults for each column. So this starts at IO_SIZE + key_length, which can be derived from the .frm header.

The format of this buffer is essentially:

```
[null_map][encoded column data]
```

Where the null_map is 1 or more bytes, with a bit per-column that can be nullable. The total number of bytes will be:

```
(null_column_count + 7 ) // 8
```

The first bit is always set and column bits start a 1 offset in the null map. If a bit is set for the current column then this indicates the the default is null (ie. DEFAULT NULL).

If a column's default is not null, then its default data will be recorded at some offset noted in the Column metadata (described elsewhere in this document). The actual data format depends on the column type. This basically breaks down into the following cases:

- integer-types - little-endian integers of 1, 2, 3, 4 or 8 bytes
- float/double - little endian IEEE 745 values
- **decimal - either ascii strings ("3.14") < MySQL 5.0, or a binary** encoding of 9 decimal digits per 4-byte big-endian integer
- timestamp - little endian integer representing seconds relative to epoch (< 5.6)
- **timestamp2 - big-endian integer representing seconds relative to epoch (5.6+);** additionally packed fractional digits, similar to the decimal format
- date/time - encodes the various components into various bits of a 3 - 8 byte integer
- char - just a string with `length` bytes (space padded)
- **varchar - length-prefix string, with the prefix being a little endian integer of** 1 to 2 bytes.

See dbsake/mysqlfrm/mysqltypes.py unpack_type_<name> method for how each datatype is actually decoded.

6.1.4 Extra data section

The "extra" section encodes some basic table properties. These include:

- CONNECTION=<name> string (used by FEDERATED)
- ENGINE=<name> strings
- PARTITION BY string
- "auto partitioned flag" (used by NDB, at least)
- WITH PARSER - fulltext parser plugin names
- Table COMMENT '...' - only if > 254 bytes

Except for the fulltext parser plugin names (which are null terminated), all of these properties are length-prefix strings. This essentially has the format:

```
[2-byte length][<connection string>]
[2-byte length][<engine name string>]
[4-byte length][<partition by clause>][null byte]
[1-byte is_autopartitioned flag]
[parser name][null_byte] for each fulltext parser plugin used
[2-byte length][<table comment string>]
```

These strings should be decoded per the table's default character set.

6.1.5 FormInfo

The .frm form info is a section consisting of 288 bytes with integers noting the length or count of elements in the table.

The start of this section can be found at offset 64 + names_len from the .frm header and the offset is a 4 byte integer. In python this would be found via

```
>> f = open('/var/lib/mysql/mysql/user.frm', 'rb')
>> f.seek(0x0004) # "names_length" documented in the .frm fileinfo header
>> names_len, = struct.unpack("<H", f.read(2)) # always 3 in modern mysql
>> f.seek(64 + names_len)
>> forminfo_offset, = struct.unpack("<I", f.read(4))
```

Here is a description of some of the more interesting fields available in the forminfo section. This is not meant to be exhaustive but merely to document the fields necessary for interpreting pertinent column metadata.

All offsets are relative to the start of the forminfo section

column_count 2 byte integer at offset 258

The number of coumns defined on this table

screens_length 2 byte integer at offset 260

How many bytes follow the forminfo section prior to the start of the column metadata

null_columns 2 byte integer at offset 282

How many nullable columns are defined in this table

names_length 2 byte integer at offset 268

Length in bytes (including delimiters) of column names

interval_length 2 byte integer at offset 274

Length in bytes (including delimiters) of the set/enum labels

comments_length 2 byte integer at offset 284

Length in bytes of the column comments

6.1.6 Column Metadata

17 bytes per column

Followed by \xff separated column names

Followed by a null byte

Followed by null terminated interval groups with each interval group consisting of interval names \xff separated.

Followed by a single string of column comments.

CHAPTER 7

Indices and tables

Index

Symbols