Data Model In Use

From Sourceforge

Source: **http://cvs.sourceforge.net/viewcvs.py/sahana/sahana-phase2/inst/mysql-dbcreate.sql?view=markup** 11/20/05

```
/**
* The central table on a person, with their associated names
* Modules: dvr, mpr, rms, or, cms
* Last changed: 27-OCT-2005 - chamindra@opensource.lk
*/
DROP TABLE IF EXISTS person_uuid;
CREATE TABLE person_uuid (
    p_uuid BIGINT NOT NULL, -- universally unique person id
    full_name VARCHAR(100), -- the full name (contains the family name)
    family_name VARCHAR(50), -- the family name
    l10n_name VARCHAR(100), -- localized version of name
    custom_name VARCHAR(50), -- extra name field as required
    PRIMARY KEY(p_uuid)
);

/**
* Many ID card numbers (or passport or driving licence) to person table
* Modules: dvr, mpr
* Last changed: 27-OCT-2005 - chamindra@opensource.lk
```

```
*/
DROP TABLE IF EXISTS identity_to_person;
CREATE TABLE identity_to_person (
    p_uuid BIGINT NOT NULL,
    serial VARCHAR(100), -- id card #, passport #, Driving License # etc
    opt_id_type VARCHAR(10), -- can be customized in the field options
table
    FOREIGN KEY (p_uuid) REFERENCES person_uuid(p_uuid)
);

/**
* Contains the Sahana system user details
* Modules: all
* Last changed: 27-OCT-2005 - ravindra@opensource.lk
*/

DROP TABLE IF EXISTS users;
CREATE TABLE users (
    p_uuid BIGINT NOT NULL,
    user_name VARCHAR(100) NOT NULL,
    password VARCHAR(100),
    PRIMARY KEY (p_uuid),
    FOREIGN KEY (p_uuid) REFERENCES person_uuid(p_uuid)
);


/**
* Main entry table as there can be multiple entries
* per person.
* Con: Provides only who entered the person & when
* Con: Does not show what was done to the entry
* Modules: dvr, mpr
* Last changed: 27-OCT-2005 - chamindra@opensource.lk
*/
/*DROP TABLE IF EXISTS person_entry;
    CREATE TABLE person_entry (
    e_uuid BIGINT NOT NULL AUTO_INCREMENT,
    entry_date TIMESTAMP,
    user_uuid BIGINT,      -- details on the user who did the data
entry
    reporter_uuid BIGINT,  -- details on the person who reported the
entry
    p_uuid BIGINT NOT NULL,
    PRIMARY KEY (e_uuid),
    FOREIGN KEY (p_uuid) REFERENCES person_uuid(p_uuid),
    FOREIGN KEY (user_uuid) REFERENCES person_uuid(p_uuid)
);*/

/**
* Details on the person's status
* Modules: dvr, mpr, or
* Last changed: 27-OCT-2005 - chamindra@opensource.lk
*/
DROP TABLE IF EXISTS person_status;
CREATE TABLE person_status (
    p_uuid BIGINT NOT NULL,
    isReliefWorker TINYINT,
```

```
    opt_status VARCHAR(10), -- missing, ingured, etc. customizable
    updated TIMESTAMP DEFAULT NOW(),
    PRIMARY KEY (p_uuid)
);

/**
* Contact Information for a person, org or camp
* Modules: dvr, mpr, or, cms, rms
* Last changed: 27-OCT-2005 - chamindra@opensource.lk
*/
DROP TABLE IF EXISTS contact;
CREATE TABLE contact (
    pgoc_uuid BIGINT NOT NULL, -- be either c_uuid, p_uuid or g_uuid
    opt_contact_type VARCHAR(10), -- mobile, home phone, email, IM, etc
    contact_value VARCHAR(100),
    PRIMARY KEY (pgoc_uuid,opt_contact_type,contact_value)
    /**only pgoc_uuid should not be primary key
    as for a person there can be several contact types
    and values (email and mobile) or multiple mobiles
    **/
);

/**
* Details on the location of an entity (person, camp, organization)
* Modules: dvr, mpr, or, cms, rms
* Last changed: 27-OCT-2005 - ravindra@opensource.lk
*/
DROP TABLE IF EXISTS location_details;
CREATE TABLE location_details (
    poc_uuid BIGINT NOT NULL, -- this can be a person, camp or
organization location
    location_id VARCHAR(20), -- This gives
country,province,district,town - based on l10n
    opt_person_loc_type VARCHAR(10), -- the relation this location has
to the person
    address TEXT, -- the street address
    postcode VARCHAR(30), -- or ZIP code
    long_lat VARCHAR(20), -- logatitude and latitude (GPS location)
    PRIMARY KEY (poc_uuid,location_id),
    FOREIGN KEY (location_id) REFERENCES location(location_id)
);


/**
* The main details on a person
* Modules: dvr, mpr,
* Created : 27-OCT-2005 - chamindra@opensource.lk
* Last Updated : 07-Nov-2005 - janaka@opensource.lk
* Note: Removed the NOT NULL Constraint on next_kin_uuid
*/
DROP TABLE IF EXISTS person_details;
CREATE TABLE person_details (
    p_uuid BIGINT NOT NULL,
    next_kin_uuid BIGINT,
    birth_date DATE,
    opt_age_group VARCHAR(10),     -- The age group they belong too
    relation VARCHAR(50),
```

```
    opt_country VARCHAR(10),
    opt_race VARCHAR(10),
    opt_religion VARCHAR(10),
    opt_marital_status VARCHAR(10),
    opt_gender VARCHAR(10),
    occupation VARCHAR(100),
    PRIMARY KEY (p_uuid),
    FOREIGN KEY (p_uuid) REFERENCES person_uuid(p_uuid)
);

/**
* Physical details of a person
* Modules: dvr, mpr, or, cms, rms
* Last changed: 27-OCT-2005 - chamindra@opensource.lk
*/
DROP TABLE IF EXISTS person_physical;
CREATE TABLE person_physical (
    p_uuid BIGINT NOT NULL,
    opt_blood_type VARCHAR(10),
    height VARCHAR(10),
    weight VARCHAR(10),
    opt_eye_color VARCHAR(50),
    opt_skin_color VARCHAR(50),
    opt_hair_color VARCHAR(50),
    injuries TEXT,
    comments TEXT,
    PRIMARY KEY (p_uuid) ,
    FOREIGN KEY (p_uuid) REFERENCES person_uuid(p_uuid)
);

DROP TABLE IF EXISTS person_missing;
CREATE TABLE person_missing (
    p_uuid BIGINT NOT NULL,
    last_seen TEXT,
    last_clothing TEXT,
    comments TEXT,
    PRIMARY KEY (p_uuid) ,
    FOREIGN KEY (p_uuid) REFERENCES person_uuid(p_uuid)
);

/**
* Not part of MPR
*/
DROP TABLE IF EXISTS person_deceased;
CREATE TABLE person_deceased (
    p_uuid BIGINT NOT NULL,
    details TEXT,
    date_of_death DATE,
    location VARCHAR(20),
    place_of_death TEXT,
    comments TEXT,
    PRIMARY KEY (p_uuid),
    FOREIGN KEY (p_uuid) REFERENCES person_uuid(p_uuid),
    FOREIGN KEY (location) REFERENCES location(location_id)
);
```

```
/**
* Person to Report (Contact person)
* Modules: dvr, mpr,
* Created : 07-Nov-2005 - janaka@opensource.lk
*/
DROP TABLE IF EXISTS person_to_report;
CREATE TABLE person_to_report (
    p_uuid BIGINT NOT NULL,
    name VARCHAR(100) NOT NULL,
    address TEXT,
    phone VARCHAR(100),
    email VARCHAR(255),
    relation VARCHAR(100),
    PRIMARY KEY (p_uuid),
    FOREIGN KEY (p_uuid) REFERENCES person_uuid(p_uuid)
);
```

From Live Sahana Site

Fields in use by MPR module on **http://cvs.opensource.lk/sahana2/** on 11/20/05

Screen: Search [Results Table] (mod=mpr&act=search)
- Picture
- Name
- Appearance
- Missing Details
- Status

Screen: Edit Missing Person Entry (mod=mpr&act=editmp)
Screen: Report a Missing Person (mod=mpr&act=addmp)
- Status
- Identity
  - Identity Card Number
  - Passport Number
  - Driving License
- Basic details
  - Full Name
  - Family Name
  - Local Name
  - Date of Birth (YYYY-MM-DD)
  - Age Group
    - Infant (0-1)
    - Child (1-15)
    - Young Adult (16-21)
    - Adult (22-50)
    - Senior Citizen (50+)
  - Gender
    - Male Female
  - Marital Status
    - Single Married Divorced
  - Religion
    - Buddhist Christian Other
  - Race
    - Tamil Other
- Contact Details
  - Address
  - Postal Code

- Home Phone
- Mobile

Physical Details
- Eye Colour
  - Black Light Brown Blue Other
- Skin Colour
  - Black Dark Brown Fair White Other
- Hair Colour
  - Black Brown Red Blond Other
- Height
- Weight
- Comments

Other Details
- Blood Type
  - AB A+ O
- Last Seen
- Last Clothing
- Comments

Reporting Person
- Name
- Relation
- Address
- Phone
- Email

Data Model For History Tracking

New Table audit_table

```
CREATE TABLE audit_table
```

Description

Track every insert, update, delete to the Sahana database.

Fields

row_id	bigint	Unique # for each row in this table
change_id	bigint	ID for each change tracked
obj_uuid	bigint	Object ID affected

obj_type	VARCHAR(4)	'per' -- Person
		  'pdv' -- Person - Disaster victim
		  'paw' -- Person - aid worker
		  'pvol' -- Person - volunteer
		  'prep' -- Person - reporting person
		  'pmis' -- Person - Missing person
		  'psu' -- Person - Sahana user
		'grp' -- Group
		'org' -- organization
		'cmp' -- Camp

obj_type_id	int	ID representing: disaster victims, aid workers, volunteers, reporting person, missing persons, Sahana users, group, camp, organization. This field is needed to help distinguish between the many types of people stored in the person_uuid table.
time_stamp	timestamp	Timestamp when
u_uuid	bigint	User making change.
change_type	VARCHAR(4)	'ins' -- Insert 'del' -- Delete 'upb' -- Update (before) 'upa' -- Update (after)
changed_table	VARCHAR(50)	Affected table
changed_field	VARCHAR(50)	Affected field
changed_value	VARCHAR(200)	New or prior value of changed field, depending on which is applicable.

Primary & Foreign Keys

```
PRIMARY KEY(row_id)

FOREIGN KEY (obj_type_id) REFERENCES obj_type(obj_type_id)
```

Note:

The following FK is not possible.

```
FOREIGN KEY (obj_uuid) REFERENCES person_uuid(p_uuid)
```

… for 2 reasons:

1. Field obj_uuid is not limited to referencing just one table. (It can reference either person_uuid, camp_uuid, or organization_uuid tables.)
2. After deletion of the referenced object, the FK will no longer be valid.

New Table obj_data

```
CREATE TABLE obj_data
```

Description

Identify all tables & fields that hold data for a specific type of object.

Fields

obj_type	VARCHAR(4)	'per' -- Person
		  'pdv' -- Person - Disaster victim
		  'paw' -- Person - aid worker
		  'pvol' -- Person - volunteer
		  'prep' -- Person - reporting person
		  'pmis' -- Person - Missing person
		  'psu' -- Person - Sahana user
		'grp' -- Group
		'org' -- organization
		'cmp' -- Camp

obj_type_id	int	ID representing: disaster victims, aid workers, volunteers, reporting person, missing persons, Sahana users, group, camp, organization. This field is needed to help distinguish between the many types of people stored in the person_uuid table.
obj_table	VARCHAR(50)	Table holding data
obj_field	VARCHAR(50)	Field holding data

Primary & Foreign Keys

Uniqueness constraint:

obj_type_id
obj_table
obj_field

Logic For Populating Audit Data Model

shn_db_insert()

```
$obj_uuid = scan input array for uuid ;

For each field in input array,

     insert into audit table, with constants:
            change_id
            obj_uuid
            obj_type
            obj_type_id
            time_stamp
            u_uuid
            change_type
```

shn_db_update()

```
$obj_uuid = scan input array for uuid ;
```

```
For each field in input array,

     get original value in table

     insert original value into audit table, with constants:
            change_id
            obj_uuid
            obj_type
            obj_type_id
            time_stamp
            u_uuid
            change_type  -- "Before update"

     insert original value into audit table, with constants:
            change_id
            obj_uuid
            obj_type
            obj_type_id
            time_stamp
            u_uuid
            change_type  -- "After update"
```

shn_db_delete()

```
$obj_uuid = scan input array for uuid ;

For each field in target table array,

     get original value in table
     (as identified in obj_data table)

     insert original value into audit table, with constants:
            change_id
            obj_uuid
            obj_type
            obj_type_id
            time_stamp
            u_uuid
            change_type  -- "Deleted"
```

Logic For Chronological View

Chronological Changes For Missing Person

Name: Abdul Gandhi
Person UUID: 123456

Time stamp	User making change	Affected table	Affected field	Type of change	Prior value	New value
2005/11/15 22:01	Anna Naan	person_status	p_uuid	Delete	123456	---
2005/11/15 22:01	Anna Naan	person_status	isReliefWorker	Delete	0	---
2005/11/15 22:01	Anna Naan	person_status	opt_status	Delete	Alive & Well	---
2005/11/15 22:01	Anna Naan	person_status	updated	Delete	2005/11/01 15:45	---
2005/11/10 10:15	Anu Una	person_status	opt_status	Update	Missing	Alive & Well
2005/11/10 10:15	Anu Una	person_status	updated	Update	2005/11/01 15:45	2005/11/10 10:15
2005/11/01 15:45	Anna Naan	person_status	p_uuid	Insert	---	123456
2005/11/01 15:45	Anna Naan	person_status	isReliefWorker	Insert	---	0
2005/11/01 15:45	Anna Naan	person_status	opt_status	Insert	---	Missing
2005/11/01 15:45	Anna Naan	person_status	updated	Insert	---	2005/11/01 15:45

Alternative implementation of "Update"

Time stamp	User making change	Affected table	Affected field	Type of change	Value
2005/11/10 10:15	Anu Una	person_status	opt_status	After Update	Alive & Well
2005/11/10 10:15	Anu Una	person_status	updated	After Update	2005/11/10 10:15
2005/11/10 10:15	Anu Una	person_status	opt_status	Before Update	Missing
2005/11/10 10:15	Anu Una	person_status	updated	Before Update	2005/11/01 15:45

```
select
      aud.time_stamp
     ,usr.full_name
     ,aud.change_type
     ,aud.changed_table
     ,aud.changed_field
     ,aud.changed_value
from
      audit_table  aud
     ,person_uuid  usr  -- Sahana user
     ,person_uuid  pdv  -- Disaster victim
where
      aud.obj_type in ('pdv','pmis')
  and aud.u_uuid    = usr.u_uuid
  and aud.obj_uuid  = pdv.u_uuid
  and pdv.full_name = $pdv_full_name
order by
      aud.time_stamp
     ,aud.change_type
     ,aud.changed_table
     ,aud.changed_field
```

Logic For Point In Time View

Point In Time View For Missing Person

Name: Abdul Gandhi
Person UUID: 123456
View date: 2005/11/12

Affected table	Affected field	Description	Value
person_status	p_uuid	Unique person ID	123456
person_status	isReliefWorker	0=not relief worker	0
person_status	opt_status	Status	Alive & Well
person_status	updated	Last updated	2005/11/01 15:45

```
Inputs:
   $view_date
   $pdv_full_name

$pdv.p_uuid = Get_ID_From_Name($pdv_full_name) ;

create array $obj_data_fields
  select
      obj_type
      obj_type_id
      obj_table
      obj_field
  from
     obj_data
  where
      obj_type = 'pdv'


step through array $obj_data_fields

   for each field…

    * start with latest view of data

    * then go chronologically backwards in table audit_table
      to find the value at $view_date

    * store the value in the $obj_data_fields array

Display contents of array
```

Appendix: Analysis of Audit Methods

Description of the 3 methods

	Method	Description
1	(Native) RDBMS-driven tracking	Functionality built into the RDBMS (in this case mySQL) that allows certain types of tracking and change control on specified tables.
2	Trigger-driven tracking	Write trigger objects for each table that allows logging of the changes that are important to the application. Scripts can be written to build these triggers automatically from the existing data model. Typical data logged: • name of affected table • name of affected field • prior value • new value • change type: insert, update, delete • time stamp • database user ID
3	Application-driven tracking	Add code at different points in the data entry process that saves the prior data before committing a change. Typical data logged could include all fields in trigger-driven tracking, plus: • application user ID • reporter ID

Analysis of the 3 methods

	Method	Pro	Con

1	(Native) RDBMS-driven tracking	• Requires least **complexity** & development time to maintain tracking functionality • Least likely to **miss a data change** from being recorded. • Makes it easiest to **roll back** data to a certain point in time.	• Can only log database **user ID**, not application-level (such as application user ID & reporter ID) • Allows least **control** on what to log and how (I'm not sure what mySQL's capabilities are yet.) • Generates highest **volumes** of data. • Possible issue with slower **performance** than application-driven method • Least useful & least customizable **reporting** capabilities.
2	Trigger-driven tracking	• Requires less **complexity** & development time than application-driven method • Allows more **control** on what is logged and how than RDBMS-driven method • Generates less **volumes** of data than RDBMS-driven method • **Reporting** capabilities may be better than RDBMS-driven method. • Less likely to **miss a data change** from being recorded than application-driven method. •	• Can only log database **user ID**, not application-level (such as application user ID & reporter ID) • Requires more **complexity** & development time than RDBMS-driven method • Allows less **control** on what is logged and how than application-driven method • Generates more **volumes** of data than application-driven method • Possible issue with slower **performance** than application-driven method • **Reporting** capabilities not as good as application-driven method. • More likely to **miss a data change** than RDBMS-driven method. • Makes it harder to **roll back** data to a certain point in time than RDBMS-driven method.

3	Application-driven tracking	• Makes it possible to log application-level **IDs** (such as application user ID & reporter ID). • Allows highest level of **control** on what is logged & how • Generates least **volumes** of data. • Least likely to have slower **performance** issues • Allows for most useful & most customizable **reporting** capabilities.	• Requires most **complexity** & development time than other methods • Most likely to **miss a data change** from being recorded. • Makes it harder to **roll back** data to a certain point in time than RDBMS-driven method.

Comparison of Existing Event Auditing in Sahana

Function & location	Pro	Con
db.inc **shn_db_insert()** **shn_db_update()** **shn_db_delete()** **There is no shn_db_update**	• Most structured data, even more than general db_exec version. • Easiest to recreate historical data. • Less chance for omission than log_event, b/c not tightly coupled w/ database activity.	• Cannot log non-database events. • Cannot log database "select" events. • More locations of code to maintain than either db_exec or log_event.
_shn_db_exec() in file db_adodb.inc	• Possibly more structured than log_event. • Can log database "select" events. • Only 1 location of code to maintain. • Less chance for omission than log_event, b/c not tightly coupled w/ database activity. • Even shn_db_insert(), update(), delete() call _shn_db_exec()	• Cannot log non-database events. • Not as structured as function (insert, update, delete) specific version.
ADODB **exec** **selectlimit** **In file 3rd\adodb\drivers\adodb-pdo.inc.php**	• Can't miss any database calls!	• Must make sure that no Sahana code directly calls these! Only through shn_db_exec() (or scn_db_select() if it's created)

shn_log_event() in file logger.inc	• Can log non-database (application) events. • Can log database "select" events. • Only 1 location of code to maintain. • Note: Logic can be added to handle database events too.	• Least structured data. • Most difficult to recreate historical data. • More chance for omission, b/c log_event not tightly coupled w/ activity.

Sequence of function calls

Functions shn_db_insert(), shn_db_update() shn_db_delete() shn_db_select() In file inc\handler_db.inc	Call functions _shn_db_exec() In file db_adodb.inc
Functions _shn_db_exec() shn_db_select() In file db_adodb.inc	Calls functions Execute() SelectLimit() In file adodb-pdo.inc.php
Functions SelectLimit() Execute() In file adodb-pdo.inc.php	Do direct calls to database

Source Code of Existing Event Auditing in Sahana

Logs simple event message

```
For example:
shn_log_event("UUAD");
```

File: logger.inc

```
<?php
/**
* Description for file
*
* PHP version 4 and 5
*
* LICENSE: This source file is subject to LGPL license
* that is available through the world-wide-web at the following URI:
* http://www.gnu.org/copyleft/lesser.html
*
* @package    Sahana - http://sahana.sourceforge.net
```

```
* @author     Pradeeper <pradeeper@opensource.lk>
* @author     Mifan <mifan@opensource.lk>
* @copyright  Lanka Software Foundation - http://www.opensource.lk
*/


// Sahana logging (auditing) engine.
// 20th Sep 2005
// by The A-Team :)

//include(actions.inc);

function shn_log_event($message)
{

     global $conf, $global;

                    $logdate= date("jS F Y");
                    $logtime= date("G:i:s");

     $entry['userid']= $_SESSION['userid'];
     $entry['userlogid']=$_SESSION['userlogid'];
     $entry['action']=$global['action'];
     $entry['modulename']=$global['module'];


                    // switch to relevent log level based on sahana config file.
```

Note: Currently level 1 & 2 appear to be identical!

```
     switch($conf['loglevel']) {

        case 1 : // normal level auditing


_shn_db_insert_array($entry['userid'],$entry['userlogid'],$message,$conf['loglevel'],$log
date,$logtime,$entry['modulename']);
            break;

                                        case 2 : // security level auditing


_shn_db_insert_array($entry['userid'],$entry['userlogid'],$message,$conf['loglevel'],$log
date,$logtime,$entry['modulename']);
            break;


                                        default :
                                                        break;
            } // end of level_catch
}

          /**
                    * insert log entry into database
                    * connection is already open
                    * values entered via adodb
          */
          function
_shn_db_insert_array($userid,$userlogid,$message_text,$loglevel,$date,$time,$module){
                    global $conf, $global;
                    $table='cre_actionlog';
                    $id=$global['db']->GenId();
                    $query="insert into $table
values($id,$userid,$userlogid,$message_text,$loglevel,$module)";
                    @$result=$global['db']->Execute($query) or die ("Error in query:
$query".$global['db']->ErrorMsg());
          }

?>
```

Logs all calls to database

File: db.inc
inc\handler_db.inc

```
/**
 * DB Insert and update Array functions
 */
function shn_db_update($arr,$table, $key){
    global $global;
    $sql = "UPDATE $table SET ";

    foreach($arr as $k => $v){
        if($v == '')
            $sql .= "$k = NULL, ";
        else
            $sql .= "$k = ".shn_db_clean($v).", ";
    }

    $sql = substr($sql,0,strlen($sql)-2);

    if($key)
        $sql .= " $key";
//@todo: check keys and error
    if($key)
        $global['db']->Execute($sql);
#    echo $sql;
}

function shn_db_insert($arr,$table){
    global $global;
    $sql = "INSERT INTO $table ";

    foreach($arr as $k => $v){
        $keys .= "$k , ";

        if($v == '')
            $values .= "'NULL', ";
        else
            $values .= shn_db_clean($v).", ";
    }

    $keys = substr($keys,0,strlen($keys)-2);
    $values = substr($values,0,strlen($values)-2);

    $sql .= "( $keys ) VALUES ( $values ) ";

    $global['db']->Execute($sql);
    #echo $sql."<hr>";
}
function shn_db_insert_phonetic($name,$pgl_uuid)
{
    global $global;
    //split the name
    $keywords = preg_split("/[\s]+/",$name);
    foreach($keywords as $keyword){
        $arr['encode1'] = soundex($keyword);
        $arr['encode2'] = metaphone($keyword);
        $arr['pgl_uuid'] = $pgl_uuid;

        //ignore if all the fields are there
        //@todo: clean
        if(! $global['db']->GetOne("SELECT * FROM phonetic_word WHERE
encode1='{$arr['encode1']}' AND encode2='{$arr['encode2']}' AND
pgl_uuid='{$arr['pgl_uuid']}'") )
            shn_db_insert($arr,'phonetic_word');
        //clear arr just incase ;-)
        $arr = null;
    }
```

```
}
```

inc\lib_database\db.inc

```
function shn_db_insert( $table, &$hash) {
function shn_db_update( $table, &$hash, &$keyname, $verbose=false ) {
function shn_db_delete( $table, $keyName, $keyValue ) {

/**
* Insert into a table. Fields are passed as an associative array.
* @param [type] $verbose
*/
function shn_db_insert( $table, &$hash) {
          $fmtsql = "insert into $table ( %s ) values( %s ) ";
          foreach ($hash as $k => $v) {
                    if (is_array($v) or is_object($v) or $v == NULL) {
                              continue;
                    }
                    $fields[] = $k;
                    $values[] = "'" . _shn_db_escape(strip_tags( $v )) . "'";
          }
          $sql = sprintf( $fmtsql, implode( ",", $fields ) ,  implode( ",", $values ) );

          if (!_shn_db_exec( $sql )) {
                    return false;
          }
    //@todo return the id
          #$id = db_insert_id();
          return true;
}

/**
* Updates into a table. Fields and 'Where fields' are passed as an associative array.
* @param [type] $verbose
* @note after where clause joins and stuff plus n
*/

function shn_db_update( $table, &$hash, &$keyname, $verbose=false ) {
          $fmtsql = "UPDATE $table SET %s WHERE %s";
          foreach ($hash as $k => $v) {
                    if( is_array($v) or is_object($v) or $k[0] == '_' ) // internal or NA
field
                              continue;

                    if ($v == '') {
                              $val = 'NULL';
                    } else {
                              $val = "'" . db_escape(strip_tags( $v )) . "'";
                    }
                    $tmp[] = "$k=$val";
          }
    foreach ($keyname as $k => $v) {
        if( is_array($v) or is_object($v) or $k[0] == '_' ) // internal or NA field
            continue;
        if ($v == '') {
            $val = 'NULL';
        } else {
            $val = "'" . db_escape(strip_tags( $v )) . "'";
        }
        $tmpkey[] = "$k=$val";
    }

          $sql = sprintf( $fmtsql, implode( ",", $tmp ) , mplode( " AND ", $tmpkey ) );
          ($verbose) && print "$sql<br />\n";
          $ret = _shn_db_exec( $sql );
          return $ret;
}

/**
* Document::db_delete()
*
```

```
* { Description }
*
*/
function shn_db_delete( $table, $keyName, $keyValue ) {
          $keyName = db_escape( $keyName );
          $keyValue = db_escape( $keyValue );
          $ret = _shn_db_exec( "DELETE FROM $table WHERE $keyName='$keyValue'" );
          return $ret;
}
```

inc\lib_logger\logger.inc

```
_shn_db_insert_array($entry['userid'],$entry['userlogid'],$message,$conf['loglevel'],$log
date,$logtime,$entry['modulename']);

$ret = _shn_db_exec( $sql );
```

File: db_adodb.inc

```
function _shn_db_exec( $sql ) {
        global $db,$enable_cache,$cache_dir;

          if (! is_object($db))
            add_error_debug(__FILE__,__LINE__, 0, "Database object does not exist");
          /* Adding the caching ability */
          if($enable_cache)
                    $qid = $db->Execute( $sql );
          else
                    $qid = $db->Execute( $sql );
          add_error_debug(__FILE__, __LINE__, 10, $sql);
          if ($msg = db_error())
        {
                #global $AppUI;
                add_error_debug(__FILE__, __LINE__, 0, "Error executing:
<pre>$sql</pre>");
                    // Useless statement, but it is being executed only on error,
                    // and it stops infinite loop.
                    $db->Execute( $sql );
                    if (!db_error())
                              echo '<script language="JavaScript"> location.reload();
</script>';
        }
        if ( ! $qid && preg_match('/^\<select\>/i', $sql) )
           add_error_debug(__FILE__, __LINE__, 0, $sql);
          return $qid;
}
```

File: 3rd\adodb\drivers\adodb-pdo.inc.php

```
function &SelectLimit($sql,$nrows=-1,$offset=-1,$inputarr=false,$secs2cache=0)
function Execute($inputArr=false)


Examples:

$sql="SELECT metadata_text.value FROM …"
$res2 = $conn->SelectLimit($sql);

$sql = "SELECT req_id,user_id …" .
$rs = $global['db']->Execute($sql);
```