

Database

- [Compare strings by case](#)
- [Force Password Change](#)
- [Debug Values](#)
- [Unix Timestamps](#)
- [Query the Grade Scale Value](#)
- [Dates before Unix Epoch](#)
- [Disk Usage by Course](#)
- [Query Block Instances and Positions](#)
- [Get a list of Site Administrators](#)
- [Antelope to Barracuda File Format](#)
- [Assigning Site Administrations through the Database](#)
- [Too Many Connections](#)
- [Repeating Rows in MySQL and MariaDB](#)
- [Error writing to database field doesn't have a default value](#)

Compare strings by case

Here's one method of comparing strings by case in MySQL. You can use the `md5` function to hash the string and compare to its lower case value.

This example is for checking for usernames that are not lower case in moodle.

```
select username, md5(username), md5(lower(username))
from mdl_user
where deleted = 0 and suspended = 0
and md5(username) != md5(lower(username));
```

If the `md5` hash when the string is lower cased is different to the current md5 hash it will be returned and indicates a mismatch.

The `md5` hash is also handy when comparing values that might have invisible characters in them.

Force Password Change

Force password change is set in the `mdl_user_preferences` table with the value `auth_forcepasswordchange`. A value of 0 means a password change is not required while a 1 means the user will be forced to change their password. This value will typically not exist at all until a user is forced to change their password at least once.

If you ever need to bulk-force password change across all users in the system (e.g. for security reasons), you can add the `auth_forcepasswordchange` option to the `mdl_user_preferences` with a value of 1 if it doesn't exist, or simply update it from 0 to 1 if it does. Remember this is only relevant to internal authentication methods such as the default `manual` authentication method.

Debug Values

The following is a list of the numeric values to set the value of the debug field (either through `$CFG->debug` in config.php or the debug value in the in the `mdl_config` table).

- 0 = `NONE`: Do not show any errors or warnings
- 5 = `MINIMAL`: Show only fatal errors
- 15 = `NORMAL`: Show errors, warnings and notices
- 6143 = `ALL`: Show all reasonable PHP debug messages
- 38911 = `DEVELOPER`: extra debug messages for developers

To change in the database, run the following SQL statement:

```
update mdl_config set value = {appropriate value} where name = 'debug';
```

This will usually also require a purge of the Moodle cache.

Unix Timestamps

As you start developing in Moodle you'll quickly find that Unix timestamps are used everywhere in fields like `timemodified`, `timecreated` etc.

It can be handy to know how to deal with these at the database level.

MySQL and MariaDB

To convert a standard date to unix timestamp format (note this won't include timezone)

```
select unix_timestamp('2019-01-01')
```

If you need the timezone (presuming your database is set to the correct timezone, otherwise you'll need to adjust `SYSTEM` to the relevant timezone).

```
select unix_timestamp(convert_tz('2019-01-01', '+00:00', 'SYSTEM'))
```

To convert a unix timestamp to standard date format (UTC timezone)

```
select from_unixtime(1546300800)
```

Remember this doesn't take into account the timezone and will return the timezone as UTC, so you might need to add/subtract your timezone offset in seconds. E.g. GMT+11 = 11 hours * 60 minutes in an hour * 60 seconds in a minute = 39,600 seconds.

```
select from_unixtime(1546300800-(11*60*60))
```

To get the current date only (no timestamp):

```
select unix_timestamp(curdate());
```

To get the current date and time

```
select unix_timestamp(now());
```

PostgreSQL

To convert a standard date to unix timestamp format (with timezone):

```
select extract(epoch from timestamp '2019-01-01')
```

To convert a unix timestamp to standard date format (with timezone):

```
select to_timestamp('1546300800');
```

To get the current date (no timestamp)

```
select trunc(extract(epoch from current_date));
```

To get the current date and time (this will include your timezone)

```
select trunc(extract(epoch from now()));
```

Query the Grade Scale Value

Grade scales are stored in the `mdl_scale` table, however they are stored as a comma-separated list in the scale column, which makes them quite difficult to query directly from the database. Because the grade is just a number (e.g. 1, 2, 3), it is far more useful to show the corresponding grade scale grade description.

For example, the grade scale might have the following value stored in the scale column (formatted for readability with index):

- `1` Mostly separate knowing,
- `2` Separate and connected,
- `3` Mostly connected knowing

So a final grade of `2` would translate to *Separate and connected* if using this grade scale.

You can use the `substring_index` function in MySQL or MariaDB to perform a look up of the corresponding grade scale value (grade description) using the grade value.

```
select
  scale,
  trim(substring_index(substring_index(scale, ',', 2), ',', -1))
from
  mdl_scale
where
  id = 1;
```

Simply, replace the index in the inner `substring_index` call with the appropriate grade value (e.g. 1, 2, or 3) for this scale and it will give you corresponding grade scale grade description.

Dates before Unix Epoch

If you use a custom user profile field with a date and want to store a date that falls before Unix Epoch (01/01/1970) (e.g. a date of birth), the data will be stored as a negative number.

For example, a date of birth of 20/11/1959 is stored as -319284000 which means 319,284,000 seconds **before** 01/01/1970.

In MySQL/MariaDB you can convert these with the following SQL which works for both scenarios (before and after epoch, 01/01/1970) making it ideal when you can have either case.

```
select date_add(from_unixtime(0), interval data second)
from mdl_user_info_field
```

You will need to also add the relevant custom user profile field id that stores the date field you are working with.

If you just want the date without the timestamp:

```
select date(date_add(from_unixtime(0), interval data second))
from mdl_user_info_field
```


Disk Usage by Course

The following query provides a summary of the disk space usage (in bytes) and number of files used by each course in the system from the `mdl_files` meta table for the data directory / file store.

This is just for the space used in the data directory (not the database) but that is usually where most of the disk usage happens (e.g. multimedia resources associated with the course).

```
select
    f.contextid,
    x.instanceid,
    c.fullname as course_full_name,
    c.shortname as course_short_name,
    sum(f.filesize) as size_in_bytes,    sum(case when (f.filesize > 0) then 1 else 0 end) as
number_of_files
from
    mdl_files f inner join mdl_context x
    on f.contextid = x.id
    and x.contextlevel = 50
    inner join mdl_course c
    on c.id = x.instanceid
group by
    f.contextid, x.instanceid
order by
    sum(filesize) desc
;
```

NOTE: context level 50 refers to courses.

Query Block Instances and Positions

The following query gives you an idea of all the moodle blocks that have been added to various pages on your site and where they are positioned:

```
select bi.*, bp.*, x.*
from mdl_block_instances bi inner join mdl_block_positions bp
on bi.id = bp.blockinstanceid
inner join mdl_context x
on x.id = bp.contextid
order by bi.blockname;
```

Very handy if you have a block that is causing you problems e.g. not fully uninstalled or visible to users.

The pagetypepattern e.g. `*, site_index, my-index, course-view-*` tells you which page types the block will be shown on, while the `pagetype` gives you the specific instances where it appears.

The context information at the end can also tell you the contextlevel (e.g. `50 = course`, `70 = module`) and the instance of that context (e.g. `x.instanceid = 2` means course id `2` if the contextlevel is `50` (course context)).

Get a list of Site Administrators

The following SQL will help you see which users are site admins in a MySQL/MariaDB database using the `find_in_set` function to query the values listed in the `mdl_config` `siteadmins` key-value.

```
select u.*  
from mdl_user u  
where  
    find_in_set(u.id, (select value from mdl_config where name = 'siteadmins'))
```

Note you need to use `find_in_set` instead of the `in` operator which doesn't handle the commas returned to separate each siteadmin's user ID.

Antelope to Barracuda File Format

During your upgrades or on the [Server Environment Checks](#) page, you might see a message like this:

`unsupported_db_table_row_format` if this test fails, it indicates a potential problem.

Your database has tables using Antelope as the file format. You are recommended to convert the tables to the Barracuda file format. See the documentation [Administration via command line](#) for details of a tool for converting InnoDB tables to Barracuda.

This message is advising you should move to the better [InnoDB Barracuda](#) format on MySQL and MariaDB.

From the MySQL documentation:

- **Antelope** is the original InnoDB file format, which previously did not have a name. It supports COMPACT and REDUNDANT row formats for InnoDB tables and is the default file format in MySQL 5.6 to ensure maximum compatibility with earlier MySQL versions that do not support the Barracuda file format.
- **Barracuda** is the newest file format. It supports all InnoDB row formats including the newer COMPRESSED and DYNAMIC row formats. The features associated with COMPRESSED and DYNAMIC row formats include compressed tables, efficient storage of off-page columns, and index key prefixes up to 3072 bytes (`innodb_large_prefix`).

To perform the table upgrades, you use the Moodle CLI and run the following commands (prefix with `sudo` if required):

Script for detection of row size problems in MySQL InnoDB tables.

By default InnoDB storage table is using legacy Antelope file format which has major restriction on database row size. Use this script to detect and fix database tables with potential data overflow problems.

Options:

```
-i, --info          Show database information
-l, --list          List problematic tables -f, --fix      Attempt to fix all tables
                    (requires SUPER privilege)
-s, --showsql       Print SQL statements for fixing of tables -h, --help      Print out
this help
```

First check for problematic tables:

```
php admin/cli/mysql_compressed_rows.php -l
```

The `-l` parameter lists tables that need compacting/compressing e.g.

mdl_data	Compact	(needs fixing)	mdl_data_fields
Compact	(needs fixing)	mdl_enrol_paypal	Compact (needs fixing)
mdl_lti	Compact	(needs fixing)	mdl_user
Compact	(needs fixing)		
mdl_user_info_field	Compact	(needs fixing)	

To fix, run the command with the `-f` fix parameter.

NOTE: this can be database intensive so avoid running during high load periods or when there are a lot of active users on the site. Ideally you should do this during an outage in [maintenance mode](#).

```
php admin/cli/mysql_compressed_rows.php -f
mdl_data          ... Compressed mdl_data_fields          ... Compressed
mdl_enrol_paypal  ... Compressed mdl_lti                    ... Compressed
mdl_user          ... Compressed
mdl_user_info_field ... Compressed
```

To confirm all is well, run the list command again or check the environments page. There is also a handy `-i` parameter to check your database platform details:

```
php admin/cli/mysql_compressed_rows.php -i
Database version:      5.7.23-log
Database name:         moodle
Database engine:       InnoDB
innodb_file_per_table: ON
innodb_file_format:    Barracuda
```

Assigning Site Administrations through the Database

The following steps can be used to add your user as a siteadmin in the database if required.

Step 1: Find your moodle user ID

```
select id from mdl_user  
where username = '{username}';
```

Step 2: Get the list of current siteadmin IDs

```
select value from mdl_config where name = 'siteadmins';
```

This gives you a list of Moodle user ids in a list that are `siteadmin`s. What you want to do is append your Moodle user ID to the end of the list (assuming it isn't there already!).

Step 3: Add your moodle user id to the list

You can do this manually, simple add `',{your_moodle_userid}'` to the text or you can use a query like this:

```
select concat(value, ',', (select id from mdl_user where username = '{username}'))from mdl_config
where name = 'siteadmins';
```

Step 4: Update the siteadmin list

The following SQL does the deed of updating the list of siteadmins so be very careful. Use the results from step 3 to perform this update. If anything goes wrong use the results from step 2 to revert the changes.

```
update mdl_config
set value = '{list_of_siteadmins_with_your_moodle_userid}'
where name = 'siteadmins';
```


Too Many Connections

When your Moodle database is overloaded you might see errors like this (MySQL/MariaDB) in your PHP error log and a similar message in your MySQL/MariaDB log.

```
mysqli::mysqli(): (HY000/1040): Too many connections in
/lib/dml/mysqli_native_moodle_database.php on line 376mysqli::mysqli(): (08004/1040): Too many
connections in /lib/dml/mysqli_native_moodle_database.php on line 376
```

This indicates you have reached your `max_connections` settings.

Check in MySQL with an appropriately privileged account e.g. `root`.

```
mysql> SELECT @@max_connections;
+-----+
| @@max_connections |
+-----+
|                200 |
+-----+
```

Also check how many are in use using:

```
mysql> SHOW STATUS LIKE 'max_used_connections';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| Max_used_connections | 16    |
+-----+-----+
```

This defined in your `my.cnf` configuration e.g. under `/etc/my.cnf` though it may not be set and you

may need to add it as a setting.

```
[mysqld]
max_connections = 200
```

NOTE: you can set this in the mysql client while it is online if you don't want to do a DB restart e.g.

`SET GLOBAL max_connections = 200;` but but aware this will be cleared next restart so make sure you also update `my.cnf`.

Repeating Rows in MySQL and MariaDB

You might come across a need to repeat a row a set number of times in your SQL. There's an easy way to do this but you first need to create a `numbers` table with just one column which is a count starting at 1 and up to whatever number you need. For this example I'll use 10, purely to save space but you would just increase this to any suitable value.

```
CREATE TABLE `numbers` (  
  `count` int(2) DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
BEGIN;  
INSERT INTO `numbers` VALUES (1);  
INSERT INTO `numbers` VALUES (2);  
INSERT INTO `numbers` VALUES (3);  
INSERT INTO `numbers` VALUES (4);  
INSERT INTO `numbers` VALUES (5);  
INSERT INTO `numbers` VALUES (6);  
INSERT INTO `numbers` VALUES (7);  
INSERT INTO `numbers` VALUES (8);  
INSERT INTO `numbers` VALUES (9);  
INSERT INTO `numbers` VALUES (10);  
COMMIT;
```

Now we have a table we can join our rows against up to the count stored in the `numbers` table. Let's take a trivial example. Say I want to repeat the rows for a user in the `user` table 10 times. Here's how you would do this:

```
select *  
from mdl_user u join numbers n  
on n.count <= 10  
where u.id = 2;
```

This will repeat the same row in the `user` table for the user with `id` equals `2` 10 times.

Error writing to database field doesn't have a default value

If you receive an error like this, it indicates that your database schema may be missing some defaults:

```
Default exception handler: Error writing to database Debug: Field 'idnumber' doesn't have a default value
```

In this example, the error occurred when creating a new user account. To confirm, you can use the XMLDB editor to check your defaults:

Site administration > Development > XMLDB Editor > Check Defaults

This will allow you to run a check on the database and pick up any missing defaults. It will also generate the database SQL to fix this. For example, the check in this scenario found 4 defaults missing on the `mdl_user` table for the columns `idnumber`, `phone1`, `phone2` and `theme`. It provided the following SQL to run on the database to fix:

```
ALTER TABLE mdl_user MODIFY COLUMN idnumber VARCHAR(255) COLLATE utf8mb4_unicode_ci NOT NULL  
DEFAULT '' after password;ALTER TABLE mdl_user MODIFY COLUMN phone1 VARCHAR(20) COLLATE  
utf8mb4_unicode_ci NOT NULL DEFAULT '' after emailstop;ALTER TABLE mdl_user MODIFY COLUMN phone2  
VARCHAR(20) COLLATE utf8mb4_unicode_ci NOT NULL DEFAULT '' after phone1;ALTER TABLE mdl_user  
MODIFY COLUMN theme VARCHAR(50) COLLATE utf8mb4_unicode_ci NOT NULL DEFAULT '' after
```

```
calendartype;
```

Once you run the SQL on the database, you can use the same functionality to confirm that the defaults are now in place.