VIEW Development Guideline

Table of Contents

[1. Development Environment 4](#_Toc395872213)

[1.1 Linux development 4](#_Toc395872214)

[1.2 Windows development 4](#_Toc395872215)

[1.3 Database environment 4](#_Toc395872216)

[2. PHP Coding Standard 4](#_Toc395872217)

[2.1 General development rules for adding new file 4](#_Toc395872220)

[2.2 General development rules for modifying code 5](#_Toc395872221)

[2.3 PHP file formatting 5](#_Toc395872222)

[2.3.1 Text Files 5](#_Toc395872223)

[2.3.2 Images 5](#_Toc395872224)

[2.3.3 HTML, XML and SGML Guidelines 5](#_Toc395872225)

[2.4 Coding style 6](#_Toc395872226)

[2.4.1 Indentation 6](#_Toc395872227)

[2.4.2 Control Structures & Braces 6](#_Toc395872228)

[2.4.3 Whitespace 7](#_Toc395872229)

[2.4.4 Parenthesis 8](#_Toc395872230)

[2.4.5 Comments 8](#_Toc395872231)

[2.4.6 ONE-TIME Code comment 8](#_Toc395872232)

[2.4.7 Function Calls 8](#_Toc395872233)

[2.4.8 PHP Code Tags 9](#_Toc395872234)

[2.4.9 Global Variables 9](#_Toc395872235)

[2.4.10 Alternatives & General Issues 9](#_Toc395872236)

[3. Comment rule 10](#_Toc395872237)

[3.1 PHPDocumentor introduce 10](#_Toc395872239)

[3.2 How to comment our code 10](#_Toc395872240)

[3.2.1 Basic syntax 10](#_Toc395872241)

[3.2.2 Summary comment 11](#_Toc395872242)

[3.2.3 Description comment 11](#_Toc395872243)

[3.2.4 A series of tags 12](#_Toc395872244)

[3.3 Useful Tags 12](#_Toc395872245)

[3.3.1 author 12](#_Toc395872246)

[3.3.2 param 12](#_Toc395872247)

[3.3.3 return 13](#_Toc395872248)

[3.3.4 version 13](#_Toc395872249)

[3.3.5 example 14](#_Toc395872250)

[3.4 Old/Unused Codes 15](#_Toc395872251)

[4. MySQL Coding Standard 15](#_Toc395872252)

[4.1 MySQL naming conventions 15](#_Toc395872255)

[4.1.1 Table Naming 15](#_Toc395872256)

[4.1.2 Field Naming 15](#_Toc395872257)

[4.2 MySQL script coding style 16](#_Toc395872258)

[4.2.1 Table Key 16](#_Toc395872259)

[4.2.2 Table Index 16](#_Toc395872260)

[4.2.3 Referencing Fixed Rows 16](#_Toc395872261)

[4.2.4 SQL Statements 17](#_Toc395872262)

[4.2.5 Not Embedding SQL in PHP 17](#_Toc395872263)

[4.2.6 Replication Safe Query 17](#_Toc395872264)

[5. Coding Performance Guideline 18](#_Toc395872265)

[5.1 General Performance Guideline 18](#_Toc395872268)

[5.1.1 Stress Test 18](#_Toc395872269)

[5.1.2 New Module Deployment 18](#_Toc395872270)

[5.1.3 Historical Data Migration 18](#_Toc395872271)

[5.2 Replication error and lagging 18](#_Toc395872272)

[5.2.1 Not insert local Global database 18](#_Toc395872273)

[5.3 MySQL performance 18](#_Toc395872274)

[5.3.1 Database Connection 18](#_Toc395872275)

[5.3.2 Dynamiccatalogue 18](#_Toc395872276)

[5.3.3 Temp table 19](#_Toc395872277)

[5.3.4 Sub query 19](#_Toc395872278)

[5.3.5 Cache table 19](#_Toc395872279)

[5.3.6 Rules of Index Usage 19](#_Toc395872280)

[5.3.7 General rules for SQL 19](#_Toc395872281)

[6. Development Environment Standard Operations/Discipline 21](#_Toc395872282)

# Development Environment

Both Linux and Windows development platforms are allowed for VIEW application development.

## Linux development

The default development environment is KDE on Fedora Core Linux, while you are free to use other Linux desktop environments. You should familiarize yourself with the following applications which are available on your desktop.

kate – the KDE advanced text editor

kdesvn – A graphical KDE subversion frontend

kompare – A graphical KDE diff frontend

svn – The command line svn program

## Windows development

NetBeans – Free IDE to do code development

Tortoise SVN – SVN client to connect to TKE SVN repository

Winmerge – another good tool to do code compilation

WinSCP/FileZilla – FTP client to connect TKE preview environment

Firebug – to debug js, monitor network as well as checking functions through web page

JMetter(or ab) – Stress Testing tool to simulate multiple concurrent users to test server load

Putty – use it to connect client server

MySQL Workbench – connect to TKE preview and RC databases

Cisco VPN – to connect to TKE internal Network

## Database environment

Each vendor should setup its own internal development environment with replication in place to simulate the same environment as what is VIEW’s live environment

# PHP Coding Standard







## PHP file formatting

### Text Files

In general all text files should use latin1 or UTF-8 encoding and should use unix newline conventions, where other encodings or newline conventions are required they may be used however their use and the reasons for such use should be documented and noted in the revision control log

### Images

The preferred raster image format is either PNG or JPEG, in some instances it may be necessary to use 8-bit PNGs rather than 24-bit PNGs in order to have transparency work correctly in Internet Explorer 6 or lower.

### HTML, XML and SGML Guidelines

All markup code should be as close to XHTML 1.0 compliant as practical, generally this means that all tag and attribute names should be in lower case and that self-closing tags should be denoted as such using the XHTML 1.0 convention of adding a trailing slash before the closing angle bracket

## Coding style

### Indentation

Logical block indentation should be performed using <TAB> characters (U+0009). The preferred tab width is 4 characters, however provided code displays correctly at other tab widths you may configure your editors tab width to any size. Where aligned indentation is desired, such as for multi-line strings, it should be performed with tabs followed by spaces, such that altering the displayed tab width only affects the depth of logical block indentation.

Examples

1: If ($yes) {

2: $SQL = "SELECT COUNT(\*)\n"

3: . " FROM table\n";

4: }

### Control Structures & Braces

Opening braces should usually be placed on the same line as their related control structure, while closing braces should usually be placed on their own line. In some cases it may be acceptable to place both the opening and closing brace, along with the statement(s) to be executed on the same line as the control structure, e.g. if you use braces for an if block any related else or else if blocks must also use braces.

Examples

1: If ($yes) {

2: print "OK";

3: } else if ($no) {

4: print "I don't think so";

5: } else {

6: print "How should I know?";

7: }

If the control structure runs onto multiple lines you should use the same rules as long function calls, but add the opening brace to the line containing the closing parenthesis of the control structure condition unless doing so reduces readability

1: function foo($bar) {

2: print "foobar";

3: }

If the control structure runs onto multiple lines you should use the same rules as long function calls, but add the opening brace to the line containing the closing parenthesis of the control structure condition unless doing so reduces readability

1: if (

2: $yes || $no || $maybe || $notsure || $idontthinkso

3: || $idontknow || $couldbe

4: ) {

5: print "What?";

6: }

Also we don’t suggest to use a assign value statement or call method in if or else if statement which is unreadable and may raise issue between languages, for example as a bad example,

if($tableChanged = $db->get("SELECT \* from `systemsetting` WHERE `DefineSymbol` = '" .$tableChangedString . "' AND `Value` =1")){

$valColumns = self::getValColumns($db,$tableName);

$suffixArr = array("en"=>"en","local"=>"local");

...

}

At least below points need to be modified in above code

1. Don’t use assign statement or call a method in if statement
2. Don’t pass a long SQL or string into a method directly, and can define a variable instead
3. Do’not combine SQL statement directly, use quote method to escape value string is necessary when pass a value a value to SQL query
4. No comment for the block

As an good example, please refer below block:

// Use SQL to check if xxxxxx

$strSql = "SELECT \* from systemsetting WHERE DefineSymbol = ". $db->quote($tableChangedString)." AND Value = 1"

$isTableChanged = $db->get($strSql)

if($isTableChanged){

$valColumns = self::getValColumns($db,$tableName);

$suffixArr = array("en"=>"en","local"=>"local");

...

}

If many statement in if, try to break statement in each line to make code is readable as below:

if($\_REQUEST['recommended\_subcontractor\_type'] == 'I'

|| $\_REQUEST['recommended\_subcontractor\_type'] == 'II'

|| $\_REQUEST['recommended\_subcontractor\_type'] == 'III'

|| $\_REQUEST['recommended\_subcontractor\_type'] == 'Scaffold'

|| $\_REQUEST['recommended\_subcontractor\_type'] == 'Hoisting'){

// update instance status and service branch status to active if meet type

$applicationOperate->updateStatus($intInstanceID,'Active');

$applicationOperate->updateServiceBrancheStatus($intInstanceID);

...

}

### Whitespace

* + Operators & operands should be separated by whitespace.
  + Parentheses should not be separated from their contents by whitespace, but should be separated from operators by whitespace.
  + Arguments should be separated from each other by whitespace between the terminating "," of each argument and the value of the next argument.
  + Control structures should be separated from their expressions by a single space.
  + There should be no space between a function name and it's parenthesis
  + Optional parameters in function declarations should not be separataed from thier defaults values or the assignment operator which assigns it.

Separation should always be by means of the space character (U+0020). In general you should always use one space the separate items, but it is acceptable and even encouraged to use more where doing so will allows for vertical alignment of parts of related statements.

Examples

1: function foo($bar, $it=-FALSE) {

2: print "foobar";

3: }

1: if ($yes || $no) {

2: $a['Test' ] = 10;

3: $a['Certification'] = 20;

4: $a['Live' ] = 30;

5: }

1: function foo($bar) {

2: $m = max ($a, $b, $c, $d);

3: }

1: switch ($variable) {

2: case ‘condition 1’ :

3: Operation 2

4: Break;

5:

6: case ‘condition 2’ :

7: Operation 2

8: Break ;

9:

10: Default :

11: Default operation

12: Break;

13 }

### Parenthesis

Parenthesis should be included anywhere where the grouping or operator precedence of an expression is unclear, regardless of whether they are strictly required. They should also be used to represent logical groups within an expression where doing so increases clarity.

Examples

1: if ($yes || ($no && $maybe)) {

2: $a['Test' ] = 10;

3: $a['Certification'] = 20;

4: $a['Live' ] = 30;

5: }

### Comments

For comment details, please refer to Comment section

### ONE-TIME Code comment

Sometimes one-time code like dirty data clearance will be send to code reviewer to review, but if no comment to point this is a one-time code, code reviewer will have concern of this kinds of code, to let code reviewers can better understand which code is one-time code, please add comment for the one-time code as below style.

“Comment symbol” >>>>>Start One-Time code: dirty data clearance

Code …

Code …

……

“Comment symbol” <<<<<End One-Time code: dirty data clearance

Note: please replace “Comment symbol” with related code language.

### Function Calls

Function calls should generally be made on a single line with no whitespace between the function name and the opening parenthesis of it's argument list. In the event that the argument list is too long to reasonably fit on a single line the opening parentheses should be placed on the same line as the function name and the arguments should be placed on one or more lines indented from the function name by a single tab (U+0009) character, in this case the closing parenthesis should be placed on it's own line at the same indentation as the function name.

Examples

1: $to = 'tester@example.com';

2: mail($to, 'A test email', 'This is a test email');

1: define\_enum(

2: 'JSTYPE\_INT',

3: 'JSTYPE\_REAL',

4: 'JSTYPE\_BOOL',

5: 'JSTYPE\_STRING',

6: 'JSTYPE\_ARRAY',

7: 'JSTYPE\_OBJECT',

8: 'JSTYPE\_HASH',

9: 'JSTYPE\_REGEX'

10: );

Also do not use “&” to a object parameter because PHP will pass the object by reference by default.

### PHP Code Tags

PHP code should always start with a complete XML Processing Instruction tag as show below, unless specifically required for a particular project short open tags of the form "<?" or "<?=" should never be used as they conflict with the ability to perform XML transformations on the document. The ASP syntax "<%...%>" should also be avoided. If code is placed on the same line as the opening PHP tag then the matching closing tag should be placed on the same line.

Examples

1: <?php

2: print "Hello World";

3: ?>

1: <?php print "Hello World"; ?>

### Global Variables

When using global variables (other than super-globals such as $\_REQUEST) inside a function you should import the necessary global into the function's scope using a global statement at the top of the function rather than referencing it via the $GLOBALS super global array wherever possible. Nothing should come below a global declaration inside a function block. Placing the global declaration at the top of the function in this way allows quick reference as to what things outside the function may affect or be changed by it's execution.

Examples

1: function foo($bar, $it=FALSE) {

2: global $myglobal;

3: print $myglobal

4: }

### Alternatives & General Issues

* + Always use print instead of echo
  + Do not treat print as a function, it is a language construct and you should not surround it's arguments in parenthesis.
  + Always use implode() instead of join()
  + Always prefer explode() to split() unless regular expressions are actually required.
  + Always use the \_once() form of include() or require() where appropriate.
  + When inlining variables in double quoted strings always use the "{$var}" format.
  + Make sure that the closing ?> of a file doesn't have any unintentional whitespace after it, especially in libraries.
  + Ensure that every argument that is passed to an external program is first passed through escapeshellarg().
  + Don’t create unnecessary loop and be careful about looping within looping
  + Should not go through double loop to display the data. E.g. for each bank, if there are different rows of attributes, should not loop through each row to find the correct row to display. The correct way to do it is to build the array with each attribute as the key: e.g. $UnitTypeData[$bankId][$ScopeofworkID or defined symbol] should be able to display the data for certain scope of work attribute directly.

# Comment rule

We suggest using the standard comment rule to comment our code, so that it is possible to general some API document for developers by some tools like PHPDocumentor.



## PHPDocumentor introduce

PHPDocumentor is the world standard auto-documentation tool for PHP. Written in PHP, PHPDocumentor can be used directly from the command-line. PHPDocumentor can be used to generate professional documentation directly from the source code of your PHP project. Support for linking between documentation, automatic class inheritance and generation of highlighted source code with cross-referencing to php general documentation are just a few of the features of PHPDocumentor.

## How to comment our code

If the file is very useful or with many API function for other modules to call, it is very important to comment well.

Every developer should try to comment their codes clearly and regularly.

The comments are about three levels:

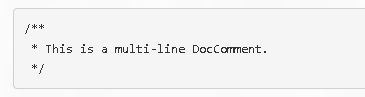
* File level
* Class level
* Function level

### Basic syntax

A Comment starts with a forward slash and two asterisks (/\*\*), which is similar to how you start a multiline comment but with an additional asterisk, and ends with an asterisk and forward slash (\*/).

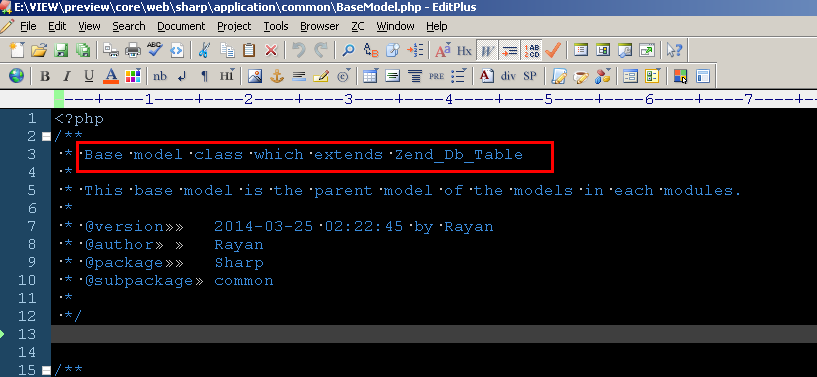
Comments may be a single line in size but may also span multiple lines, in which case each line must start with an asterisk. It is customary, and recommended, to align the asterisks vertically when spanning multiple lines.

For example:



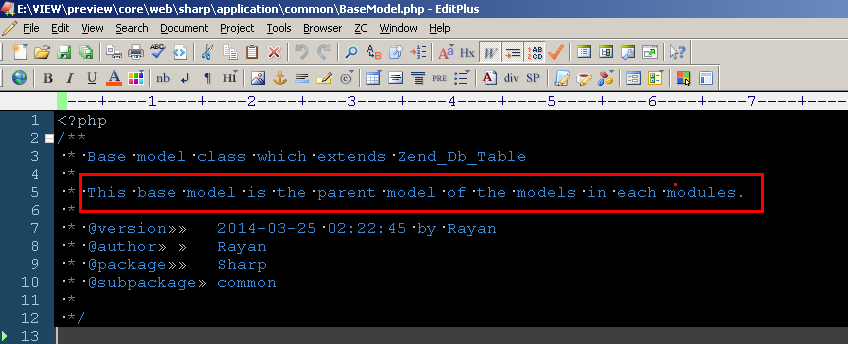
### Summary comment

A short piece of text, usually one line, providing the basic function of the associated element.



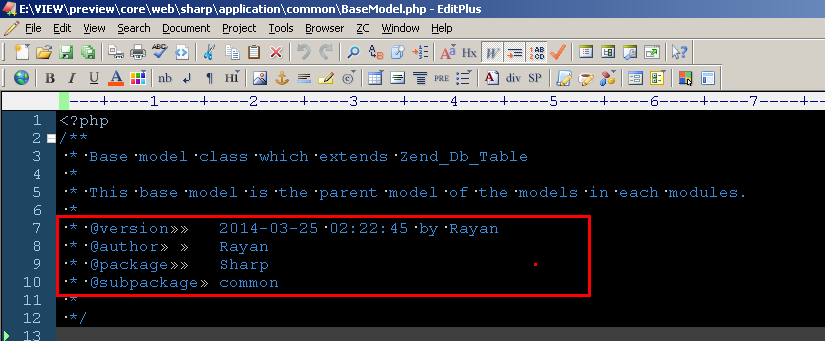
### Description comment

An optional longer piece of text providing more details on the associated element’s function. This is very useful when working with a complex element.



### A series of tags

These provide additional information in a structured manner. With these tags you can link to other elements, provide type information for properties and arguments, and more.



## Useful Tags

### author

The @author tag can be used to indicate who has created.

Syntax:

@author [name] [<email address>]

Example:

/\*\*

\* @author My Name

\* @author My Name <my.name@example.com>

\*/

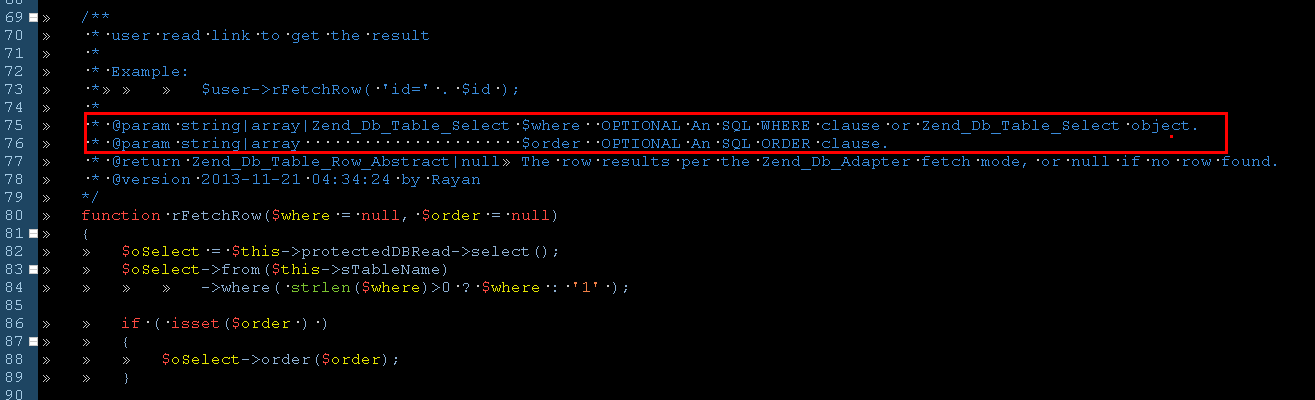
### param

The @param tag is used to document a single argument of a function or method

Syntax:

@param [Type] [name] [<description>]

Example:



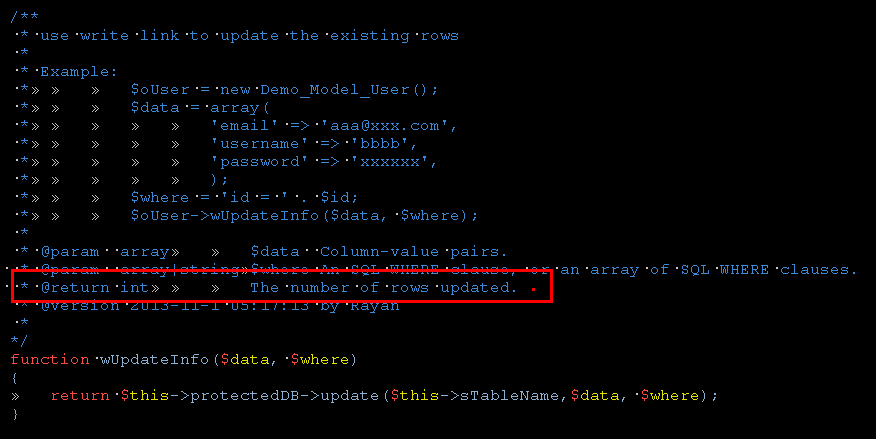
### return

The @return tag is used to document the return value of functions or methods.

Syntax:

@return [Type] [<description>]

Example:



### version

The @version tag indicates the current version

Syntax:

@version [<vector>] [<description>]

Example:

/\*\*

\* @version 1.0.1

\*/

class Counter

{

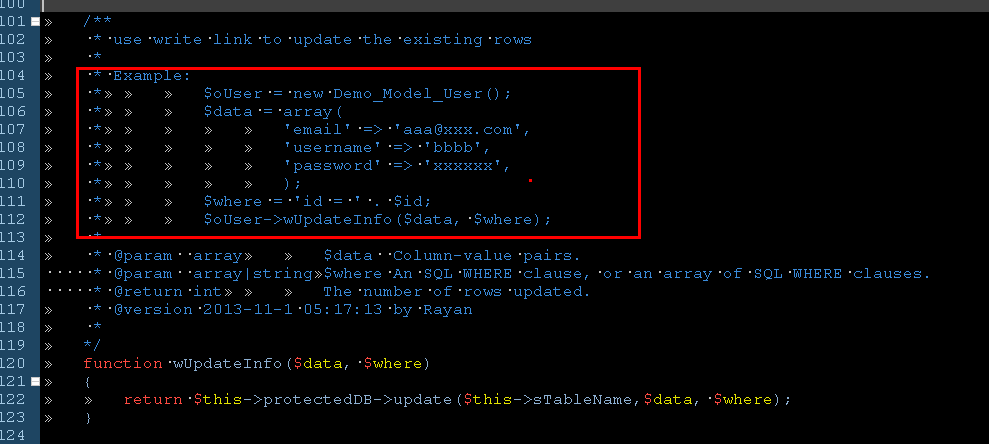
<...>

}

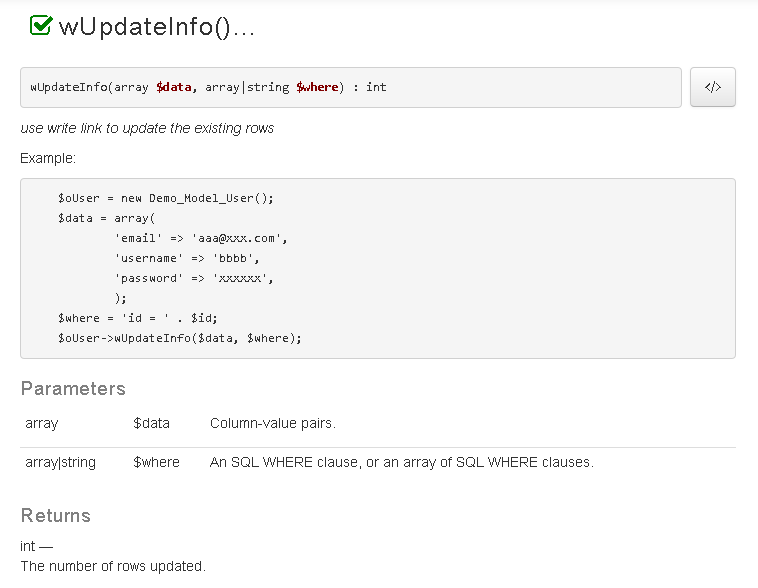
### example

The default example syntax is not friendly for us to use, so please add the example to the Description comment if needed.

Example:



Then the API page will show as below:



## Old/Unused Codes

Unless special reason, old/Unsed codes should be removed rather than comment. For the special case, use standard comment rule to indicate it.

# MySQL Coding Standard







## MySQL naming conventions

### Table Naming

In general table names should always be the lower case singular form of the object which a row in them represents

It is not allow any permanent tables or views with temp/tmp/back/backup/bak in the name.

Join tables should be named <table1>\_<table2> with the "\_" implying the join

### Field Naming

Field names should generally use CamelCase with the exception of the artificial primary key field "id"

## MySQL script coding style

### Table Key

Primary, Foreign and Unique keys should be specified after all of the fields in the CREATE statement rather than as modifiers in the field definition itself.

Whenever an artificial primary key is used the field should simply be called "id" in the owner table, but <tablename>id when used as a foreign key.

Examples

1: CREATE TABLE accountstatus

2: id int unsigned NOT NULL AUTO\_INCREMENT,

3: Name varchar(64) NOT NULL,

4: DefineSymbol varchar(32) NOT NULL,

5: PRIMARY KEY (id),

6: UNIQUE KEY (DefineSymbol)

7: );

8:

9: CREATE TABLE account (

10: id int unsigned NOT NULL AUTO\_INCREMENT,

11: Username varchar(255) NOT NULL,

12: FirstName varchar(255) NOT NULL,

13: Surname varchar(255) NOT NULL,

14: AccountStatusID int unsigned NOT NULL,

15: PRIMARY KEY (id),

16: UNIQUE KEY (Username),

17: FOREIGN KEY AccountStatusID REFERENCES accountstatus(id)

18:);

### Table Index

Making use of proper index: avoid the case: miss index, duplicated index, improper index etc.

Tables which provide reference data only for one other table (or concept) should have their name prefixed with the table which refers to them.

### Referencing Fixed Rows

Any table where rows are referenced by static id's from code should define a unique varchar column "DefineSymbol" by which the row can be referenced, by registering the table in the "definesymbols" table the initialization code will build all the necessary constants to refer to the id using the constant "<PREFIX>\_<DEFINESYMBOL>"

Examples

1: CREATE TABLE accountstatus

2: id int unsigned NOT NULL AUTO\_INCREMENT,

3: Name varchar(64) NOT NULL,

4: DefineSymbol varchar(32) NOT NULL,

5: PRIMARY KEY (id),

6: UNIQUE KEY (DefineSymbol)

7: );

8:

9: INSERT INTO accountstatus VALUES (DEFAULT, 'Active', 'ACTIVE' );

10: INSERT INTO accountstatus VALUES (DEFAULT, 'Not Active', 'INACTIVE');

11:

12: INSERT INTO definesymbols VALUES ('accountstatus', 'ACCOUNT');

13:

14: -- 'Not Active' can now be written ACCOUNT\_INACTIVE in the php code

### SQL Statements

Each clause of the SQL statement should be right aligned to a common point determined by the length of the longest clause identifier, with all parts of that clause left aligned to a point one character to the right of the common alignment point. Simple SQL statements may be expressed in a single line.

JOINs should always be spelled out explicitly not abbreviated to "," and conditions of the JOIN (as opposed to filters) should be placed in the ON condition wherever possible rather than in the WHERE clause.

Examples

1: SELECT e.id, e.Name, e.Date, v.Name AS Venue, COUNT(\*) AS Bookings

2: FROM event

3: JOIN booking b ON e.id = b.EventID

4: LEFT JOIN venue v ON e.VenueID = v.id

5: WHERE (b.UserID = @ME OR b.UserID IS NULL)

6: GROUP BY e.id

7: ORDER BY e.Date DESC

8: LIMIT 10

### Do not add DEFINER attribute when create store procedure, function, trigger…

Stored programs and views are defined prior to use and, when referenced, execute within a security context that determines their privileges. These privileges are controlled by their DEFINER attribute, and, if there is one, their SQL SECURITY characteristic.

DEFINER can be explicited defined as below which will become default owner of the store procedure. Because the specified account will be removed when member leave which may lead to unknown risk. And we don’t have a policy to check the DEFINER account, so we don’t suggest to define the attribute in statemement explicitly.

According to current VIEW deployment process, member in team will use specified account to do deployment, and this account will become default owner of the store procedure. It will reduce risk for this.

So in summary, the code same as below:

**Non Compliance Sample:**

1: CREATE DEFINER = 'admin'@'localhost' PROCEDURE p1()

2: SQL SECURITY DEFINER

3: BEGIN

4: UPDATE t1 SET counter = counter + 1;

5: END;

Compliance Sample:

1: CREATE PROCEDURE p1()

2: BEGIN

3: UPDATE t1 SET counter = counter + 1;

4: END;

For details of DEFINER attribute, please refer mysql help with below link

<http://dev.mysql.com/doc/refman/5.0/en/stored-programs-security.html>

### Not Embedding SQL in PHP

Please try avoid embedding SQL into PHP. Such statements should be expressed using the database library API rather than directly as SQL.

### Replication Safe Query

Insert / update/ delete operation must be performed in the Master database base table instead of slave database.

Insert / update /delete operation must be performed in Master database base table but cannot insert into a view which point to another database.

When inserting into Dynamiccatelogue, NEXTXID function call must be invoked first and separated from the insert statement

e.g. SET @variable := NEXTXID();

INSERT INTO dynamiccatalogue (xid, val\_en) VALUES (@variable, 'foo');

Every developed SQL script must be verified in development environment with N-Master replication in place, especially for SQL operating on global database, e.g. dynamiccatalogue, customer, customeraddress, etc.

# Coding Performance Guideline



## General Performance Guideline

### Stress Test

Stress Test plan and Test need to be executed to check the performance of module or functions that will be released

### New Module Deployment

All new large module or large enhancements should have different menus or system setting to turn on country by country.

### Historical Data Migration

All the new module or enhancement should consider the historical data migration.

## Replication error and lagging

### Not insert local Global database

Do not insert local Global database, instead, move the function to APAC with correct permission setup, APAC site will be central point of inserting data into Global database when developing/design new functions in country site.

## MySQL performance

### Database Connection

There will have more VIEW DB servers with load-balanced in live environment, please make sure follow existing code and please don’t try initiate DB connections everywhere

### Dynamiccatalogue

New developed functions need to store its own multiple languages into its own table instead of still writing into dynamiccatagolue table

Never write update of dynamiccatalogue without any where condition. For example, following SQL is strictly prohibited.

UPDATE dynamiccatalogue    SET Val\_cn= REPLACE(Val\_cn,'\\\\',''),Val\_en= REPLACE(Val\_en,'\\\\',''),Val\_hk= REPLACE(Val\_hk,'\\\\',''),Val\_ko= REPLACE(Val\_ko,'\\\\',''),Val\_th= REPLACE(Val\_th,'\\\\',''),Val\_cntw= REPLACE(Val\_cntw,'\\\\',''),Val\_vn= REPLACE(Val\_vn,'\\\\','');

For new function that you are working on, SQL\_C\_Static and SQL\_X\_Static shall be not used.

### Temp table

Try eliminating the usage of tmp table, if necessary, using memory table instead to avoid more I/O

### Sub query

Avoid writing long query with subquery. Try to use separate query to replace the subquery and create proper indexes on the separate query to improve performance.

Try eliminate sub-query with limit 1

Whenever possible, please use “Explain” to check what is the query plan and use “Profile” to see what was happening behind

### Cache table

Try to avoid cache table if possible. The cache table should be for multiple users. Should try to avoid create cache table for report of one person.

Avoid using “delete cache table”, try use truncate, as it is faster than delete statement

Consider update cache base on update one instead of refresh whole table, i.e. update set \*\*\* employee\_cache where id = 3), then the cache will be updated more efficiently

### Time Zone

In some scenario, we need to insert current data time into database just like field last update time. in this scenario, normally developer will use now method in MySQL. but as VIEW system support lots countries with different time zone, and even different in same country. So as rule, in VIEW development, we forbid to use Now method, instead to use a PHP constant “BRANCHNOW” to add the current time into database.

### Rules of Index Usage

Index can speed up the query when table has lots data, but meanwhile, index will occupies more disk space, and also will slow down the update, insert and delete action. So we may need to take of index creating. Below is some rules of index usage.

1. The table's primary key, foreign key must have an index;

2. The amount of data exceeds 300, the table should have index;

3. The field is often used in “JOIN” clause needs to be indexed

4. The field is often used in “Where” clause needs to be indexed, especially the big table

5. The index should be built on the high selectivity of the field;

6. the index should be built on a small field, even for large text fields and long field, not to build an index;

7. create a composite index requires careful analysis, should try to consider using a single-field index instead:

8. do not create too many indexes if this table has much frequent access/action

9. delete unwanted index, to avoid a negative impact on the implementation plan;

### General rules for SQL

1. If you don’t need all rows, add **limit** at the end
2. If you don’t need all columns, select [specific columns]. However, if need to share same piece of codes, you may need to use select \*
3. Use Explain or Explain Extended to see the sql execution plan
4. Try NOT to use Optimizer hints, FOR UPDATE and LOCK IN SHARE MODE
5. Try to ensure that any group by or order by expression refers only to coumns from a single table, so that MySQL can try to use an index for that operation.
6. Use statement in count method like below:

SELECT COUNT(color = ‘blue’ OR NULL) AS blue, COUNT(color = ‘red’ OR NULL) AS red FROM ITEMS

1. Use tool pt-query-advisor
2. Some Optimization samples
3. Delete sql

**BAD SQL**:

Delete FROM messages WHERE created < DATE\_SUB(NOW(), INTERVAL, 3 MONTH)

**Good SQL:**

rows\_affected = 0;

do {

rows\_affected = do\_query (

“DELETE FROM message WHERE created < DATE\_SUB(NOW(), INTERVAL, 3 MONTH) LIMIT 10000”)

} while rows\_affected >0

Deleting 10,000 rows at a time is typically a large enough task to make each query efficient, and a short enough task to minimize the impact on the server (transactional storage engines might benefit from smaller transactions). It might also be a good idea to add some seep time between the **delete** statements to spread to load over time and reduce the amount of time locks are held.

1. Do not use this kind of subquery

SELECT \* FROM sakila.film WHERE film\_id IN (

SELECT film\_id FROM sakila.film\_actor WHERE actor\_id =1)

“IN” statement suffering a bad performance, for this case, we can use where or left join to replace the “IN” statement.

1. Put limit inside of Union

**BAD SQL:**

(SELECT first\_name, last\_name FROM sakila.actor ORDER BY last\_name)

UNION ALL

(SELECT first\_name, last\_name FROM sakila.customer ORDER by last\_name)

LIMIT 20;

**Good SQL:**

(SELECT first\_name, last\_name FROM sakila.actor ORDER BY last\_name LIMIT 20)

UNION ALL

(SELECT first\_name, last\_name FROM sakila.customer ORDER by last\_name LIMIT 20)

LIMIT 20;

1. Large query-set optimize

**BAD SQL**

SELECT file\_id, description FROM sakila.film ORDER BY title LIMIT 50, 5;

**GOOD SQL:**

SELECT film.file\_id, film.description FROM sakila.film

INNER JOIN (

SELECT film\_id FROM sakila.film ORDER BY title LIMIT 50, 5) AS lim using(film\_id)

1. **Do add join condition when use join statement**

**If developer did not use condition on join, database will calculate a largest result collection which will be with poorest performance.**

**BAD SQL**

SELECT …

FROM

repair r JOIN

repair\_timesheet rt JOIN

timesheet t JOIN

…

WHERE

rt.RepairID = r.id

AND rt.TimesheetID = t.id

…

r.IsScheduled = 0

**GOOD SQL:**

SELECT …

FROM

repair r JOIN

repair\_timesheet rt ON rt.RepairID = r.id JOIN

timesheet t ON rt.TimesheetID = t.id JOIN

…

WHERE

…

r.IsScheduled = 0

**As a result, after changed SQL, the execution time of query decreased from 20 seconds to 0.2 seconds.**

# General Standards of VIEW Developing

## General development rules for adding new file

To let all logic of the code more clearly and easier to maintain, please do not use one file to do all things, should be separate to multiple files, the files includes css segment, javascript logic, SQL logic, business logic handling and html.

Rules of adding new files:

1. Css code must be placed to .css file. (For old VIEW modules, put file at core\web\css. For MNI module location, put files at core\web\nimod\public\css. For new template, put files at core\web\sharp\css)
2. Js code must be placed to .js file (For old VIEW modules, put files at core\web\js. For MNI module put files at core\web\nimod\public\js. For new template, put files at core\web\sharp\js). All js file from external website, should be downloaded and put to VIEW server. The protocol should be specified in src.
3. business logic must be placed to .lib.php file(For old VIEW modules: core\sys\libs\logic. For new template, the business logic that used by other modules should be placed in \core\web\modules\[actual module name like servicesales]\controllers\ServiceController.php)
4. SQL logic must be placed to .php file (Data access layer, only fetch data, not include business logic handling)
5. Front page only put html and include those files, and only contains simply page render logic.

## General development rules for modifying code

### Effected Function Estimation

To minimize bug and provide good quality of system, it is suggested that before change any public variant, functions, or fields, the developer need to check dictionary to see whether it will affect other functions.

### Server Side Validation

Normally we will add validation in frontend UI by java script for some format like number, date time format. but in some scenario, we may forget scenario like copy and past, so we do suggest to do server side validation to make sure the data quality, and avoid potential issue. It can be ajax way to improve user experience.

### General development rules for date format

The date format of VIEW system can be set in System Settings of System Administration module, and all dates on the system will be displayed in specified format.

Do note that in order to keep date format in consistency among MYSQL, PHP, JQUERY, JS DATE REGEX, PHP DATE REGEX and JS DATE REGEX, we have defined seris date format constants base on specified date format in setting page as below.

|  |  |  |  |
| --- | --- | --- | --- |
| Constant Name | Where Used | Is Visible on Page | Comment |
| DATE\_FMT | System Settings page for selection | Y | Visible to users for date format selection on System Settings page |
| MYSQL\_DATE\_FMT | SQL for operating tables | N | Invisible to users, format automatically updated based on **DATE\_FMT** selected |
| PHP\_DATE\_FMT | PHP code for formatting date | N | Invisible to users, format automatically updated based on **DATE\_FMT** selected |
| JQUERY\_DATE\_FMT | JQuery for formatting date | N | Invisible to users, format automatically updated based on **DATE\_FMT** selected |
| DATE\_REGEX\_JS | JS for date regular expression | N | Invisible to users, format automatically updated based on **DATE\_FMT** selected |
| DATE\_REGEX\_PHP | PHP code for date regular expression | N | Invisible to users, format automatically updated based on **DATE\_FMT** selected |

Developers should follow the rules below on manipulating data format under different circumstances:

* Use **MYSQL\_DATE\_FMT** when selecting, updating, inserting, and deleting tables.

e.g. “DATE\_FORMAT(cf.ExpectedDate, '" . **MYSQL\_DATE\_FMT** . "') AS ExpectedDate,”

* Use **PHP\_DATE\_FMT** when formatting date in PHP code.

e.g. “strftime(**PHP\_DATE\_FMT**, strtotime($date)))”

* Use **JQUERY\_DATE\_FMT** when formating date in JQuery.

e.g. “JSDateFmt ="<?php echo **JQUERY\_DATE\_FMT** ?>";

$("#startdate").datepicker({

dateFormat:JSDateFmt,

showAnim: "slideDown",

showOn: 'button',

buttonImage: 'image/calendar.gif',

buttonImageOnly: true,

changeMonth: true,

changeYear: true });”

* Use **DATE\_REGEX\_JS** when applying regular expression in JS.

e.g. “function validate\_date(datestr, req) {

// required is true if undefined

if (req == false) {

var regex = /^(<?php print **DATE\_REGEX\_JS**; ?>)?$/;

} else {

var regex = /^<?php print **DATE\_REGEX\_JS**; ?>$/;

}

if(regex.test(datestr)) {

return true;

}

return false;

}”

* Use **DATE\_REGEX\_PHP** when applying regular expression in PHP code.

e.g. “if (ereg('^' . **DATE\_REGEX\_PHP** . '$', $data['fromdate'])) {

$dbdata['FromDate'] = $db->quote(mysql\_date($data['fromdate']));

}”

### Multiple language fields and dynamic catalogue

In original design of VIEW system, all fields maybe with multiple language will be stored in dynamic catalogue table. An XID will be generated for reference, like a project name or address. It can help solve multiple language for dynamic contents. But according to business volume, the dynamic catalogue is with about twenty million records which is affect performance too much, also will make the sql statement complex, while less fields of multiple language in dynamic catalogue is used.

A new refactor project started to remove dynamic catalogue dependency for existing table. so please follow up below rules when referred a dynamic catalogue fields during development:

1. Forbid to add new XID fields in existing table or new table

For fields really need multiple language support, suggest to add two fields in table like ProjectName\_en, ProjectName\_local.

1. Obey the new style of code for existing XID fields if they have been refactored

You can ask your leader for list of existing tables that has been refactored

## General development rules for Database Change

VIEW is big system with complicated architecture, and always in an ever-evolving process in both coding and database. Currently soruce code is managed by SVN. For database, we want to keep a clean empty database always, and by which we can start a new environment easily. In VIEW we called it Skeleton DB which contains below two actual database in Dev database server:

* skel\_global
* tk\_skel\_country\_

All DDL (create/alter table, etc…) operation need to be executed in above 2 DBs by **developer** to keep Skeleton DB in updates. There will be a check item in deployment check list to tracking this.

For DDL operations, please refer to:

<http://www.orafaq.com/faq/what_are_the_difference_between_ddl_dml_and_dcl_commands>

* CREATE - to create objects in the database
* ALTER - alters the structure of the database
* DROP - delete objects from the database
* TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
* COMMENT - add comments to the data dictionary
* RENAME - rename an object

## Other Standards

1. Explicitly add UTF-8 Encoding for Popup window

Because some popup doesn’t inherit any layout, so for such scenario, if the popup contains some local language like Chinese, Korean, etc…, messy code will be displayed. To avoid this, UTF8 encoding is required to add explicitly as below in header.

<meta http-equiv="Content-Type" content="text/html; charset=**UTF-8**" />

Also, for such do need case to cover local language scenario.

1. **Year list of “From Year” and “To Year” in reports**

We have a setting called “Market Size Entry Start From Year”, and default set to 2010 currently. So options of “From Year” and “To Year” can be calculated with below logic.

* First option of year list is from default setting, and is 2010 currently.
* Last option of year list is year of 2 fiscal year later, so it will depend on current month.

For example,

* If current month is 2014/09 which belongs to 13-14 fiscal year, the last option will be 2016
* If current month is 2014/10 which belongs to 14-15 fiscal year, the last option will be 2017.

# Development Environment Standard Operations/Discipline

1. All codes should be committed to dev/dev2/rc svn before it can be updated to the dev/dev2/rc respectively
2. Please note when we use the scp command, we should use the limit bandwidth parameter when copy over the WAN to prevent impact to the network. The following command which limit the bandwidth to around 1-3Mbps should be used.
   1. connect to India: scp -l 1000
   2. connect to China: scp -l 3000
   3. connect to other: scp -l 2000
3. Fill the dev2 and RC svn revision number in ITCM ticket.
4. Don’t run the large SQL on Dev, Test report writer in Dev2.
5. Need to consider data migration when fix bug or enhancement.
6. Do take care about performance when coding; like don’t access DB several times in a large loop; Try to user memory cache or other method to reduce DB access.
7. To update dev2 code, writing data in READ DB is strictly prohibited and will cause replication problem. Remember to review the architecture diagram to understand the structure before work on dev2 enviroment.
8. When changing in one place will affect some other functions, please document it in the ITCM or SP. Also make sure testers and reviewers aware of it.
9. When doing the validation like comparing dates, make sure the code get the current site date format so that validation will be correct.
10. Both BA and Developer should be with sense for master data duplication check when writing functional spec and coding.

For a bad example, we found that there are lots duplicated service provider in VIEW Service which caused by end user who registered duplicated records.

As a best practice, as system designer we need to add name duplication check at least when add or edit a new service provider.

1. In some project or change request, we may need to develop some background jobs including shell, php script or python, or something else. For those jobs, we may need to follow up below rules:
   1. The job script need to be put SVN for control, the path is: **\core\sys\bin**
   2. When deploy, a specified path in server for routine execution and permission group should obey the deployment checklist.

Other suggestions to developer are:

1. Try to understand ur develop target from business logical aspect, not only spec or instruction.
2. Do think what is the purpose, what user want, stand at user standpoint.