Data Manipulation API

This API defines the functionality to access database tables. All the functionality is accessed via the global `$DB` object, which is an instance of the `moodle_database` class and is initialised – including opening the connection – from the settings in the ***/config.php*** file. Apart from the usual querying, updating, inserting, and deleting records in database tables, the API provides many compatibility public methods, as Moodle supports several databases.

Cross-DB Compatibility

Use the SQL compatibility methods to ensure the SQL is compatible with the supported databases (see https://docs.moodle.org/dev/Data_manipulation_API#Cross-DB_compatibility) e.g.

- `get_in_or_equal()` - Constructs 'IN()' or '=' SQL fragment and returns an SQL snippet and a parameter array to specify if a value is IN the given list of items.
- `sql_concat()` - Returns a snippet to do CONCAT between the field names passed and with `sql_concat_join()`, using passed in character(s) as the separator.
- `sql_isempty()` and `sql_isnotempty()` - Returns the snippet to query whether one field is empty or not.
- `sql_like()` and `sql_like_escape()` - Returns 'LIKE' snippet of a query and/or escape the LIKE special characters such as '_' or '%'.
- `sql_substr()` - Returns the proper snippet used to extract substrings.

Handling Query Results

It is possible to control how database query results are handled by using the strictness parameter which is supported by several methods that expect a single record.

- Passing the constant `IGNORE_MISSING` as the strictness parameter will return a boolean `false` if a record is not found or generate a debugging message if multiple records are found (this is the default behaviour).
- Passing `MUST_EXIST` will, instead, throw an exception.
- There is another constant, `IGNORE_MULTIPLE`, that will only return the first of multiple records, but this is not recommended and may be deprecated in the future.

SQL Parameters

Just about all the data manipulation methods expect a placeholder (`$params`) parameter, which is an array of values to fill placeholders in SQL statements. Using placeholders avoids issues with SQL-injection and invalid SQL quoting and helps maintain cross-DB compatible code. There is support for two types of placeholders.

- The `SQL_PARAMS_QM` replaces the '`?`' placeholders in the SQL and is single-dimensional. It must contain the same number of items as placeholders in the SQL and replacement is sequential.
- The `SQL_PARAMS_NAMED` is a multidimensional array, with the keys matching the placeholders in the SQL. The SQL placeholders are the key names with a '**:**' (colon) prefix.

Stipulating Conditions

There are several ways to specify the conditions for SQL queries.

- The simplest form is a multidimensional array `$conditions`, with the field name as the key and the value as the field's value. The array items are joined with the `AND` statement in the `WHERE` clause, so all conditions must be met to generate a result.
- A string containing the `WHERE` conditions (`xxx_select()`)
- A full SQL command (`xxx_sql()`).

Methods

- Getting a single record - `get_record()`, `get_record_select()`, `get_record_sql()`
- Getting multiple records - `get_records()`, `get_records_select()`, `get_records_sql()`, `get_records_list()`
- Getting data as key/value pairs in an associative array - `get_records_menu()`, `get_records_select_menu()`, `get_records_sql_menu()`
- Counting records that match the given criteria - `count_records()`, `count_records_select()`, `count_records_sql()`
- Checking if a given record exists - `record_exists()`, `record_exists_select()`, `record_exists_sql()`

Methods (cont)

- Getting a particular field value from one record - `get_field()`, `get_field_select()`, `get_field_sql()`
- Getting field values from multiple records - `get_fieldset_select()`, `get_fieldset_sql()`
- Setting a field value - `set_field()`, `set_field_select()`
- Deleting records - `delete_records()`, `delete_records_select()`
- Inserting records (objects) - `insert_record()`, `insert_records()`
- Updating records (objects) - `update_record()`

Record Sets

Where there is a large number of records returned, it is best to use the recordset methods that return an iterator which must be closed when no longer required.

- `get_recordset()`
- `get_recordset_select()`
- `get_recordset_sql()`
- `get_recordset_list()`

A list of all the query and manipulation methods and their expected parameters is in the relevant handout in this lesson

Transactions

- `start_delegated_transaction()`, which starts the transaction and returns a transaction object. Delegated database transactions can be nested; the outermost transaction will only be committed if all the nested delegated transactions commit successfully. Any rollback in the nested transactions will roll back all the transactions.
- `allow_commit()` method will commit the transaction
- `rollback()` method expects the exception as the parameter

Links & Handout

Data manipulation API
https://docs.moodle.org/dev/Data_manipulation_API

Reminder: Handout of DB Manipulation Functions in this lesson.