



The Handbook

Codex

http://codex.wordpress.org/Using_Alternate_Databases

Version Date

8 August 2005

Using Alternative Databases

Right now, WordPress only supports MySQL databases. A number of people have requested support for other databases; particularly **PostgreSQL**, but also some others. This page is an effort to summarize the [previous discussion](#) (<http://wordpress.org/support/topic.php?id=549&page=1>) on the topic, and to get a solid roadmap for supporting more databases in WordPress.

Port vs. Integration

An excellent [PostgreSQL port](#) (<http://wordpress-pg.sourceforge.net/>) of WordPress exists, written by Keenan Tims. While this is a great first step and sufficient for some needs, a port is not desirable to many users who wish to have the latest security and features available from main WordPress branch. A better solution would be to integrate support for alternative databases into WordPress. This, however, will take a concerted effort from the developers to write queries which port easily, and to get together a good abstraction layer for the database.

Challenges

1. Current codebase is very MySQL-centric. While WordPress does use the [ezSQL](#) (<http://php.justinvincent.com/>) class to implement database calls, this cannot properly be called an abstraction layer. Differences in sql syntax implementations among different databases (whether literals are surrounded by quotes, limit query syntax, case sensitivity, [etc.](#) (http://phplens.com/lens/adodb/tips_portable_sql.htm)) are not cared for by ezSQL, which simply offers a generic "query" call. This means that a large number of queries would have to be rewritten, a large task.
2. Database driven plugins depend on the current implementation, and also use mysql-centric code. Even if the wordpress code were to be rewritten to support a full database abstraction layer, any sudden shift in the database implementation will likely break others' plugins.
3. Current standard database abstraction layers ([ADODB](#) (<http://adodb.sourceforge.net/>), [Pear DB](#) (<http://pear.php.net/package/DB>)) are very large and complex, they would represent dependencies as large as or larger than wordpress itself. This could reduce WordPress' portability and ease of installation.
4. Installation/upgrades would be made more complicated with the addition of multiple database support. The current upgrade and installation code is entirely MySQL-centric, and difficult to generalize.

Solutions

Status Quo

Maintain the current development direction. Very little importance is placed on database independence by the WordPress core developers at the moment, so a fork is necessary to

support other databases.

Pros:

- This approach is already in place for (at least) PostgreSQL support, existing code needs only maintenance
- Cooperation from the WordPress development team is a non-issue
- Performance optimizations can be done with no concern for other databases

Cons:

- This approach is very costly for the port developers. Each release of WordPress must be scoured for incompatibilities, ported, and tested independently.
- No user-support for users of alternative databases from the WordPress team

Extend to support PostgreSQL

Extend ezSQL to include any necessary abstractions to make MySQL and PostgreSQL work equally well. This would be a matter of including both ezSQL's Postgres abstraction and their MySQL abstractions in the installation, as well as adding necessary methods to generalize between the two (i.e., a [limitQuery method](http://pear.php.net/manual/en/package.database.db.db-common.limitquery.php) (<http://pear.php.net/manual/en/package.database.db.db-common.limitquery.php>), perhaps others). Rewrite SQL queries throughout to be PostgreSQL/MySQL compatible, and to use custom extensions to ezSQL to handle incompatibilities where they exist. Users would set which database they were using in wp-config.php and not have to think about it further.

Pros:

- This solution is probably the minimal solution to properly support both PostgreSQL and MySQL, and keeps a low footprint and no large dependencies.
- Legacy plugins dependent on MySQL could still function so long as they are used on a MySQL installation.
- Use of the existing WordPress-Pg port code may make this approach even easier.

Cons:

- The solution doesn't easily allow support for other databases, and is generally substantially less clean than using a full abstraction layer of some sort.
- Some queries simply cannot be expressed in a database agnostic way (e.g. getting the next value for auto_increment/sequence), which will result in a potentially nasty mix of abstraction and "direct access".
- Abstracting error handling will be hard (will it? --KT).
- Some developers (both of the core project and of plugins) may not keep their changes in sync for both databases.
- Database queries would continue to be scattered all over the code.

WordPress-specific Database Abstraction Layer

Convert all queries (and blocks of queries) in WordPress to function calls, or more appropriately, object methods, to get that relevant information out of the database. Support for new databases can be added by adding code for that database to these functions. Optimizations for specific databases when getting certain blocks of information can be added to the relevant functions. (For example, fetching the front page posts and their

comment count can be done in one query on databases that support subqueries vs. requiring at least the number of posts shown + 1 queries with legacy MySQL.) Legacy plugins should still function when MySQL is being used.

Pro:

- All the code and queries that would need tweaking to support a new database would be in one file rather than being shotgunned all over the codebase as is presently the case.
- Optimizations for specific databases would be easier to implement and maintain (using, for example, subselects); this could offset any overhead from the functional calls. Optimization benefits may exist for newer versions of MySQL as well.
- Future schema changes and cleanup wouldn't require changing as many files.
- No new dependencies besides database support for the database the user is utilizing.
- Schema improvements that could benefit all databases would be easier to implement.
- Ability to support arbitrary storage backends like RSS feeds, text files, or an install of a different piece of blogging software (not just databases). This will also make the upgrade and migration code a lot cleaner and easier.

Con:

- The extra function calls and their packaging of data will create a small amount of additional overhead.
- This option will be a fair amount of work and will require developers of the main project and plugins to become familiar with it.
- Code for the various databases may fall out of sync without some diligence on the part of developers.
- Accessing the database data in new ways in the future will require additional functions to be written.
- Additional forethought is required in figuring out what database accesses to group together to reach the full optimization potential.

Full Database Abstraction Layer

Convert all queries in WordPress to function with a database abstraction layer such as ADOdb or Pear DB.

Pro:

- Cleanly implements database usage in the most portable means possible, allowing use of WordPress with any database supported by the chosen abstraction layer.

Con:

- These tools create large dependencies and break legacy plugins that depend on mysql specific functions.
- Large generic database (and persistent object) abstraction implementations tend not to be exceedingly fast and may not use optimizations that could benefit specific databases.
- This option will be **a lot** more work (likely less than implementing a WP-specific layer, however) and will require developers of the main project and plugins to be familiar with the abstraction layer chosen.
- Much diligence will be necessary for developers to ensure that database-specific

code doesn't make it into queries despite the abstraction layer, and this goes for plugin developers too.

- Some functionality can't be abstracted easily with ADODB or (to a much greater extent) PEAR; if Wordpress uses this functionality, it may have to be reimplemented application-side, which would be a performance hit

Money?

A number of people (see the previous [discussion thread](http://wordpress.org/support/topic.php?id=549) (<http://wordpress.org/support/topic.php?id=549>)) had proposed donations to pay for development of one of these choices for alternative database support in WordPress.