Mechanisms for tracking DB schema changes [closed]

Asked 16 years, 4 months ago Modified 3 years, 1 month ago

Viewed 46k times Part of PHP Collective

139

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 12 years ago.

What are the best methods for tracking and/or automating DB schema changes? Our team uses Subversion for version control and we've been able to automate some of our tasks this way (pushing builds up to a staging server, deploying tested code to a production server) but we're still doing database updates manually. I would like to find or create a solution that allows us to work efficiently across servers with different environments while continuing to use Subversion as a backend through which code and DB updates are pushed around to various servers.

Many popular software packages include auto-update scripts which detect DB version and apply the necessary changes. Is this the best way to do this even on a larger scale (across multiple projects and sometimes multiple environments and languages)? If so, is there any existing code out there that simplifies the process or is it best just to roll our own solution? Has anyone implemented something similar before and integrated it into Subversion post-commit hooks, or is this a bad idea?

While a solution that supports multiple platforms would be preferable, we definitely need to support the Linux/Apache/MySQL/PHP stack as the majority of our work is on that platform.

PHP php mysql database svn migration

Share

Improve this question

Follow

edited Aug 29, 2008 at 19:09

Polsonby

22.9k ● 19 ● 60 ● 74

asked Aug 4, 2008 at 21:31

pix0r

31.3k ● 18 ● 87 ● 102

20 Answers

Sorted by: Highest score (default)

In the Rails world, there's the concept of migrations, scripts in which changes to the database are made in

57

Ruby rather than a database-specific flavour of SQL. Your Ruby migration code ends up being converted into the DDL specific to your current database; this makes switching database platforms very easy.

For every change you make to the database, you write a new migration. Migrations typically have two methods: an "up" method in which the changes are applied and a "down" method in which the changes are undone. A single command brings the database up to date, and can also be used to bring the database to a specific version of the schema. In Rails, migrations are kept in their own directory in the project directory and get checked into version control just like any other project code.

This Oracle guide to Rails migrations covers migrations quite well.

Developers using other languages have looked at migrations and have implemented their own language-specific versions. I know of **Ruckusing**, a PHP migrations system that is modelled after Rails' migrations; it might be what you're looking for.

Share Improve this answer

Follow

edited May 2, 2016 at 9:04

Andreas

2,827 ● 26 ● 30

answered Aug 4, 2008 at 22:45

Joey deVilla

8,473 ● 2 ● 30 ● 15

1 Ruckusing FTW - we adapted it to our db system and are quite happy with it. – Piskvor left the building Oct 7, 2009 at 15:12

It is now located at github: github.com/ruckus/ruckusing-migrations – user409460 Jun 29, 2015 at 23:00

50

We use something similar to bcwoord to keep our database schemata synchronized across 5 different installations (production, staging and a few development installations), and backed up in version control, and it works pretty well. I'll elaborate a bit:

To synchronize the database structure, we have a single script, update.php, and a number of files numbered 1.sql, 2.sql, 3.sql, etc. The script uses one extra table to store the current version number of the database. The N.sql files are crafted by hand, to go from version (N-1) to version N of the database.

They can be used to add tables, add columns, migrate data from an old to a new column format then drop the column, insert "master" data rows such as user types, etc. Basically, it can do anything, and with proper data migration scripts you'll never lose data.

The update script works like this:

> - Connect to the database.

> - Make a backup of the current database (because stuff *will* go wrong) [mysqldump].
> - Create bookkeeping table (called _meta) if it doesn't exist.
> - Read current VERSION from _meta table. Assume 0 if not found.
> - For all .sql files numbered higher than VERSION, execute them in order
> - If one of the files produced an error: roll back to the backup
> - Otherwise, update the version in the bookkeeping table to the highest .sql file executed.

Everything goes into source control, and every installation has a script to update to the latest version with a single script execution (calling update.php with the proper database password etc.). We SVN update staging and production environments via a script that automatically calls the database update script, so a code update comes with the necessary database updates.

We can also use the same script to recreate the entire database from scratch; we just drop and recreate the database, then run the script which will completely repopulate the database. We can also use the script to populate an empty database for automated testing.

It took only a few hours to set up this system, it's conceptually simple and everyone gets the version numbering scheme, and it has been invaluable in having the ability to move forward and evolving the database design, without having to communicate or manually execute the modifications on all databases.

Beware when pasting queries from phpMyAdmin though! Those generated queries usually include the database name, which you definitely don't want since it will break your scripts! Something like CREATE TABLE `mydb`.`newtable`(...) will fail if the database on the system is not called mydb. We created a pre-comment SVN hook that will disallow .sql files containing the `mydb` string, which is a sure sign that someone copy/pasted from phpMyAdmin without proper checking.

Share Improve this answer

Follow

edited Dec 5, 2013 at 15:03

Dinah

54k ● 30 ● 134 ● 149

answered Aug 22, 2008 at 14:44

rix0rrr

10.2k ● 6 ● 47 ● 49

How did you handle collisions? Multiple developers changing the same element in the DB, for instance a stored procedure? This can happen if you're working at the same on the same branch, or you have two development lines going (two branches) – Asaf Mesika Nov 20, 2010 at 15:04

Collisions were very rare; the only thing that happened really is that two people would try to create the same N.sql file. Of course, the first one wins and the second one is forced to

rename to the next highest number and try again. We didn't have the database versioning on a branch, though. – rix0rrr Nov 25, 2010 at 13:34

12

My team scripts out all database changes, and commits those scripts to SVN, along with each release of the application. This allows for incremental changes of the database, without losing any data.

To go from one release to the next, you just need to run the set of change scripts, and your database is up-to-date, and you've still got all your data. It may not be the easiest method, but it definitely is effective.

Share Improve this answer

Follow

answered Aug 5, 2008 at 19:56

Brandon Wood

5,337 ● 4 ● 39 ● 31

1 how do you script out all changes? – Smith May 23, 2017 at 21:02

10

The issue here is really making it easy for developers to script their own local changes into source control to share with the team. I've faced this problem for many years, and was inspired by the functionality of Visual Studio for Database professionals. If you want an open-source tool with the same features, try this: http://dbsourcetools.codeplex.com/ Have fun, - Nathan.

answered Jul 7, 2009 at 13:26
Nathan Rozentals

If you are still looking for solutions : we are proposing a tool called neXtep designer. It is a database development environment with which you can put your whole database under version control. You work on a version controlled repository where every change can be tracked.

When you need to release an update, you can commit your components and the product will automatically generate the SQL upgrade script from the previous version. Of course, you can generate this SQL from any 2 versions.

Then you have many options : you can take those scripts and put them in your SVN with your app code so that it'll be deployed by your existing mechanism. Another option is to use the delivery mechanism of neXtep : scripts are exported in something called a "delivery package" (SQL scripts + XML descriptor), and an installer can understand this package and deploy it to a target server while ensuring structural consistency, dependency check, registering installed version, etc.

The product is GPL and is based on Eclipse so it runs on Linux, Mac and windows. It also support Oracle, MySQL and PostgreSQL at the moment (DB2 support is on the way). Have a look at the wiki where you will find more

detailed information : http://www.nextep-softwares.com/wiki

Share Improve this answer

Follow

edited Oct 25, 2021 at 11:37

Rizwan

103 ● 4 ● 24

answered Oct 25, 2010 at 5:46

Christophe Fondacci

141 ● 1 ● 2

Looks interesting. Does it have command-line interface as well, or is one planned? – Piskvor left the building Oct 25, 2010 at 10:46

8

Scott Ambler produces a great series of articles (and co-authored a book) on database refactoring, with the idea that you should essentially apply TDD principles and practices to maintaining your schema. You set up a series of structure and seed data unit tests for the database. Then, before you change anything, you modify/write tests to reflect that change.

We have been doing this for a while now and it seems to work. We wrote code to generate basic column name and datatype checks in a unit testing suite. We can rerun those tests anytime to verify that the database in the SVN checkout matches the live db the application is actually running.

As it turns out, developers also sometimes tweak their sandbox database and neglect to update the schema file in SVN. The code then depends on a db change that hasn't been checked in. That sort of bug can be maddeningly hard to pin down, but the test suite will pick it up right away. This is particularly nice if you have it built into a larger Continuous Integration plan.

Share Improve this answer

Follow

answered Aug 29, 2008 at 4:51

Sam McAfee

10.1k ● 16 ● 62 ● 65

7

Dump your schema into a file and add it to source control. Then a simple diff will show you what changed.

Share Improve this answer

Follow

answered Aug 6, 2008 at 16:59

deadprogrammer

11.3k ● 24 ● 75 ● 85

1 The dump has to be in SQL, like a mysqldump, Oracle's dumps are binary. – Osama Al-Maadeed Oct 20, 2008 at 8:20

7 There is also a more fundamental problem with schema diffing. How do you differentiate a column drop + add from a column rename. The answer is simple: you can't. This is the reason why you need to record the actual schema change operations. – psp Aug 1, 2010 at 7:24

The diff will show that the one column is gone, while the other appeared (unless they have the same name), and most of the time it's enough. Scripting every schema change is a

good way to go, of course: in Drupal this is handled by a special hook, for instance. – deadprogrammer Aug 3, 2010 at 20:05

6

K. Scott Allen has a decent article or two on schema versioning, which uses the incremental update scripts/migrations concept referenced in other answers here; see http://odetocode.com/Blogs/scott/archive/2008/01/31/11710.aspx.

Share Improve this answer

Follow

answered Aug 29, 2008 at 5:11

Rob

48.4k • 5 • 75 • 93

5

If you are using C#, have a look at Subsonic, a very useful ORM tool, but is also generates sql script to recreated your scheme and\or data. These scripts can then be put into source control.

http://subsonicproject.com/

Share Improve this answer

Follow

answered Aug 4, 2008 at 22:47

Dan

29.4k • 44 • 151 • 209

Seems to be a dead URL as of this point in time.
– Mark Schultheiss Feb 14, 2013 at 17:54

5

I've used the following database project structure in Visual Studio for several projects and it's worked pretty well:

Database

> Change Scripts
>
> > 0.PreDeploy.sql
> >
> > 1.SchemaChanges.sql
> >
> > 2.DataChanges.sql
> >
> > 3.Permissions.sql
>
> Create Scripts
>
> > Sprocs
> >
> > Functions
> >
> > Views

Our build system then updates the database from one version to the next by executing the scripts in the following order:

> 1.PreDeploy.sql

> 2.SchemaChanges.sql
>
> Contents of Create Scripts folder
>
> 2.DataChanges.sql
>
> 3.Permissions.sql

Each developer checks in their changes for a particular bug/feature by appending their code onto the end of each file. Once a major version is complete and branched in source control, the contents of the .sql files in the Change Scripts folder are deleted.

Share Improve this answer

Follow

answered Aug 8, 2008 at 18:31

tbreffni

5,132 ● 5 ● 32 ● 30

5

We use a very simple but yet effective solution.

For new installs, we have a metadata.sql file in the repository which holds all the DB schema, then in the build process we use this file to generate the database.

For updates, we add the updates in the software hardcoded. We keep it hardcoded because we don't like solving problems before it really IS a problem, and this kind of thing didn't prove to be a problem so far.

So in our software we have something like this:

```
RegisterUpgrade(1, 'ALTER TABLE XX ADD XY CHAR(1) NOT NULL;');
```

This code will check if the database is in version 1 (which is stored in a table created automatically), if it is outdated, then the command is executed.

To update the metadata.sql in the repository, we run this upgrades locally and then extract the full database metadata.

The only thing that happens every so often, is to forget commiting the metadata.sql, but this isn't a major problem because its easy to test on the build process and also the only thing that could happen is to make a new install with an outdated database and upgraded it on the first use.

Also we don't support downgrades, but it is by design, if something breaks on an update, we restored the previous version and fix the update before trying again.

Share Improve this answer

Follow

edited Jan 21, 2014 at 6:33

Satish Sharma

9,625 ● 6 ● 30 ● 52

answered Aug 8, 2008 at 18:21

Fabio Gomes

5,992 ● 12 ● 62 ● 79

It's kind of low tech, and there might be a better solution out there, but you could just store your schema in an SQL script which can be run to create the database. I think you

5

can execute a command to generate this script, but I don't know the command unfortunately.

Then, commit the script into source control along with the code that works on it. When you need to change the schema along with the code, the script can be checked in along with the code that requires the changed schema. Then, diffs on the script will indicate diffs on schema changes.

With this script, you could integrate it with DBUnit or some kind of build script, so it seems it could fit in with your already automated processes.

Share Improve this answer

Follow

edited Oct 25, 2021 at 11:38

Rizwan

103 ● 4 ● 24

answered Aug 4, 2008 at 22:28

Mike Stone

44.6k ● 30 ● 114 ● 140

Yeah, that's pretty much what we have in place right now. Unfortunately that doesn't give us an easy way to modify existing databases -- the SQL script generated by mysqldump assumes you're creating the table from scratch (or overwriting a table if it exists). We need something a bit more high-tech because it needs to apply a sequence of ALTER TABLE statements to the database, and in order to do that properly it needs to be aware of the current state of the database. – pix0r Aug 4, 2008 at 22:32

4

I create folders named after the build versions and put upgrade and downgrade scripts in there. For example, you could have the following folders: 1.0.0, 1.0.1 and 1.0.2. Each one contains the script that allows you to upgrade or downgrade your database between versions.

Should a client or customer call you with a problem with version 1.0.1 and you are using 1.0.2, bringing the database back to his version will not be a problem.

In your database, create a table called "schema" where you put in the current version of the database. Then writing a program that can upgrade or downgrade your database for you is easy.

Just like Joey said, if you are in a Rails world, use Migrations. :)

Share Improve this answer

Follow

answered Aug 5, 2008 at 4:36

Louis Salin

541 ● 1 ● 5 ● 8

3

For my current PHP project we use the idea of rails migrations and we have a migrations directory in which we keep files title "migration_XX.sql" where XX is the number of the migration. Currently these files are created by hand as updates are made, but their creation could be easily modified.

Then we have a script called "Migration_watcher" which, as we are in pre-alpha, currently runs on every page load

and checks whether there is a new migration_XX.sql file where XX is larger than the current migration version. If so it runs all migration_XX.sql files up to the largest number against the database and voila! schema changes are automated.

If you require the ability to revert the system would require a lot of tweaking, but it's simple and has been working very well for our fairly small team thus far.

Share Improve this answer

Follow

answered Aug 23, 2008 at 12:58

Eric Scrivner

1,849 ● 1 ● 19 ● 23

3

Toad for MySQL has a function called schema compare that allows you to synchronise 2 databases. It is the best tool I have used so far.

Share Improve this answer

Follow

answered Feb 5, 2011 at 11:08

Petah

46k ● 30 ● 159 ● 214

3

I like the way how Yii handles database migrations. A migration is basically a PHP script implementing `CDbMigration`. `CDbMigration` defines an `up` method that contains the migration logic. It is also possible to implement a `down` method to support reversal of the migration. Alternatively, `safeUp` or `safeDown` can be used

to make sure that the migration is done in the context of a transaction.

Yii's command-line tool `yiic` contains support to create and execute migrations. Migrations can be applied or reversed, either one by one or in a batch. Creating a migration results in code for a PHP class implementing `CDbMigration`, uniquely named based on a timestamp and a migration name specified by the user. All migrations that have been previously applied to the database are stored in a migration table.

For more information see the Database Migration article from the manual.

Share Improve this answer

Follow

answered Jun 25, 2011 at 13:18

Ton van den Heuvel

10.5k ● 6 ● 45 ● 89

3

Try db-deploy - mainly a Java tool but works with php as well.

- http://dbdeploy.com/
- http://davedevelopment.co.uk/2008/04/14/how-to-simple-database-migrations-with-phing-and-dbdeploy.html

Share Improve this answer

Follow

answered Jan 19, 2012 at 1:52

cwash

4,245 ● 5 ● 46 ● 53

3

I would recommend using Ant (cross platform) for the "scripting" side (since it can practically talk to any db out there via jdbc) and Subversion for the source repository. Ant will allow you to "back up" your db to local files, before making changes.

1. backup existing db schema to file via Ant
2. version control to Subversion repository via Ant
3. send new sql statements to db via Ant

Share Improve this answer

Follow

edited Oct 25, 2021 at 11:38

Rizwan

103 • 4 • 24

answered Sep 17, 2008 at 16:54

Cruz

86 • 2

2

IMHO migrations do have a huge problem:

Upgrading from one version to another works fine, but doing a fresh install of a given version might take forever if you have hundreds of tables and a long history of changes (like we do).

Running the whole history of deltas since the baseline up to the current version (for hundreds of customers databases) might take a very long time.

Share Improve this answer

answered Mar 12, 2011 at 14:15

Follow

Edson Medina
10.2k ● 4 ● 42 ● 52

0

There is a command-line mysql-diff tool that compares database schemas, where schema can be a live database or SQL script on disk. It is good for the most schema migration tasks.

Share Improve this answer

Follow

answered Nov 4, 2009 at 19:43

stepancheg
4,276 ● 2 ● 36 ● 38