Database Migrations

Imagine operating 600 remote database servers across four time zones. They might as well be on a desert island, and digitally speaking, they are. If a database administrator needed to apply changes by hand, he would have to travel to hundreds of locations.

In such circumstances, one option would be to get the database design exactly right before the first release, and then never change it again. There may still be a few people who think that's possible, but certainly none of them were on my team. We expected and even counted on change at every level, including the database.

Another option would be to send release notes out to the field. The studio managers always called the service desk for a verbal walkthrough when they executed the installs. Perhaps we could include SQL scripts in documents on the release CDs for them to type in or copy-and-paste. The prospect of dictating any command that starts with, "Now type `mysqladmin -u root -p`..." gives me cold sweats.

Instead, we decided to automate database updates. Ruby on Rails calls these "database migrations," but in 2005 it wasn't a common technique.

Updates as objects

The studio server defines a bean called a database updater. It keeps a list of database update objects, each representing an atomic change to the database. Each database update knows its own version and how to apply itself to the database.

At startup time, the database updater checks a table for the current version of the database. If it doesn't find the table, it assumes that no updates exist or have been applied. Accordingly, the very first update bootstraps the version table and populates it with one row. That single row contains a version number and a lock field. To avoid concurrent updates, the database updater first updates this row to set the lock field. If it cannot, then it assumes some other machine on the network is already applying updates.

We used this migration ability to apply some simple changes and some sophisticated ones. One of the simple ones just added indexes to a couple of columns that were affecting performance. One of the updates that made us really nervous changed all the table types from MyISAM to InnoDB. (MyISAM, the default MySQL table type, does not support transactions or referential integrity. InnoDB does. If we had known that before our first release, we could have just used InnoDB in the first place.) Given that we had deployed databases with production data, we had to use a sequence of "alter table" statements. It worked beautifully.

After a few releases had gone out to the field, we had about 10 updates. None of them failed.

Regular exercise

Every time we run a build, we reset the local development database to version zero and roll forward. That means we exercise the update mechanism dozens of times every day.