Autopatch

Autopatch

We've switched to using Autopatch to handle updates to our database tables in production servers. Currently we store database changes in sql scripts, which must then be manually applied by hand or by an installer when an installation is updated. This is error prone and creates extra work in each installer. Rather, Autopatch will maintain a table of database version numbers for each tool and the lams core, and all we need do is supply a directory of update scripts/classes; autopatch will handle the rest.

See http://autopatch.sourceforge.net for more documentation and other info; http://www.tacitknowledge.com/licenses-1.0 for the license.

How it works

Autopatch maintains a 'patches' table to store each web app's current database version. Whenever a web app starts up, Autopatch compares the database version stored in the database (aka patch level) with the version required by the web app. If older, it will apply the necessary scripts/classes to update the database and increment the database version.

How we use it

Install time patch level

In each release of lams we set the patch level at install time.

For the core, this is done in lams_common/db/insert_types_data.sql:

```
-- initialise db version
INSERT INTO patches VALUES ('lams', 13, NOW(), 'F');
```

For the tools, this is done by the tool deployer in tool/db/db_version_insert.sql:

```
INSERT INTO patches VALUES ('@signature@', '@tool_version@', NOW(), 'F');
```

As you may have seen, the lams core version number is an arbitrary number; this is the simplest way without having to second guess what the version number of the next release will be. The tool's database version is the same as the tool's tool_version number.

Startup time updates

In order to update schemas of older installations, we need to assume a starting point - LAMS 2.1. Database changes from 2.1 onwards are handled by Autopatch.

For the core, database update scripts go in lams_common/src/java/org/lamsfoundation/lams/dbupdates. This is simply a file containing SQL statements, but it must be named according to its patch level:

```
patch0013_updateFrom21.sql
```

In this example, when Autopatch executes the statements in this file, it will update the patch level to 13. 'updateFrom21' is a human readable description of the contents of the file. Whenever there is a new release, the install time patch level (set in insert_types_data.sql) should match the highest patch level contained in this directory.

If you need to put your database changes in a separate file, remember to increment the patch level, as each patch update must a separate file. Also remember to update the install time patch level in insert_types_data.sql, otherwise your script may get run when it doesn't need to.

For the tools, database update scripts go in tool/src/java/org/lamsfoundation/lams/<tool>/dbupdates. The only difference to the above is that the version number is bigger as it uses the yyyymmdd date format.

Java patches

Patches can also be written as java classes - they need to extend the MigrationTaskSupport class. Patch0011Alter21Integration.java This is an example of a patch implemented as a java class - it's a converted version of an actual patch used to update from 2.0.4 to 2.1.

Stored Procedures / Functions

Stored procedures cannot be included in a script with other items. Autopatch only handles the stored procedure if it is in a patch file all by itself. Do not include the usual autocommit and foriegn_key calls - that will cause the patch to fail. Have exactly one stored procedure and comments and nothing else. Examples of how stored procedures have been used are in lams_forum, patches 20150224.sql to 20150228.sql (see https://github.com/lamsfoundation/lams/commit/2ff38cdf504d51a2590b113023d885069d32645e#diff-3cff6d0dc09dc0cca85733428c2e6a08).

Do not use "DELIMITER $$" as that is notation for a SQL Client. Just put a ";" in the place of the delimiter. So "END$$" becomes "END;"

Problems with current patch levels

1. We want a new patch level whenever we install a server in between versions.
2. When branches include patches, we have to bump up the highest patch number on head.

To solve these, we're changing our patch numbers to resemble version release numbers. i.e. the next patch to update to 2.4 will be 02040000. This is in the format AABBCCDD:

- AA - major lams version
- BB - major release number
- CC - minor release number
- DD - sequential patch number

When installing a server in between versions, we should create a new patch file for db updates from that point on. This makes it easy to update that server in future. i.e. 2.4 is in development and the current patch number is 02040000; a new server is installed showcasing a 2.4 feature; we'd then create a new patch with number 02040001.

When merging back to the release branch, we only want the db changes that are fixes, and not the ones that are to do with new features. See examples below.

Use case: 2.3.2 release, no db changes

Head contains new features in patch 18 and patch 02040000 while the release branch is at 17. We don't include patch0018.sql in the release branch; 2.3.2 is released without any db modifications. When the 2.3.2 server is updated to 2.4, patch 18 will run, as well as 02040000.

Use case: 2.3.3 release, there are db changes

The release branch is still at 17, but there are statements inside patch02040000.sql on head that need to be part of 2.3.3. We move these to a new file on head and merge it to the release branch: patch02030300.sql. Note patch02040000.sql is NOT merged back to the release branch. This is important, as there can be no duplication here or else the server will run into problems when it is updated to 2.4.

Use case: installing server in between versions

2.3.3 is released and the release branch is at patch level 02030300. Head is at 02040000. A new server is installed showcasing a new feature in 2.4. At this point we can create a new empty patch file, patch02040001.sql. We can then let autopatch take care of updates to this server when 2.4 is released.

How it's implemented

For the LAMS core, Autopatch is invoked as a servlet listener (in order to be called before any Spring beans that will cache from the database):

web.xml

```
...

<context-param>
    <param-name>migration.systemname</param-name>
    <param-value>lams</param-value>
</context-param>
<context-param>
    <param-name>migration.databasetype</param-name>
    <param-value>mysql</param-value>
</context-param>
<context-param>
    <param-name>migration.patchpath</param-name>
    <param-value>org/lamsfoundation/lams/dbupdates</param-value>
</context-param>
<context-param>
    <param-name>migration.datasource</param-name>
    <param-value>java:jdbc/lams-ds</param-value>
</context-param>

...

<listener>
    <listener-class>com.tacitknowledge.util.migration.jdbc.WebAppJNDIMigrationLauncher</listener-class>
</listener>
<listener>
        <listener-class>
                org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>
```

For the tools, Autopatch is invoked as a Spring bean on application startup:

web.xml

```
...

<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
                                ...

                                classpath:/org/lamsfoundation/lams/tool/chat/dbupdates/autopatchContext.xml

                                ...
        </param-value>
</context-param>

...
```

autopatchContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
        <!-- Autopatch service -->
        <bean id="autopatchService" init-method="patch" class="com.tacitknowledge.util.migration.jdbc.
AutoPatchService">
                <property name="systemName" value="lachat11"/>
                <property name="databaseType" value="mysql"/>
                <property name="patchPath" value="org/lamsfoundation/lams/tool/chat/dbupdates" />
                <property name="dataSource" ref="dataSource"/>
        </bean>
</beans>
```

Apart from that, we just need to make sure that the Spring context file and SQL scripts are included in the jar when it is built:

```
        <target name="compile-java" depends="clean-build" description="compile java sources">
                <javac srcdir="${src.java.dir}" compiler="modern" destdir="${build.classes.java}" deprecation="
on" debug="on">
                        <compilerarg value="-Xlint:unchecked"/>
                        <classpath>
                                <path refid="project.classpath" />
                        </classpath>
                </javac>
                <copy overwrite="yes" todir="${build.classes.java}/${package}">
                        <fileset dir="${src.java.dir}/${package}">
                                <include name="*.xml" />
                                <include name="*.properties" />
                                <!-- Autopatch -->
                                <include name="dbupdates/*.sql" />
                                <include name="dbupdates/autopatchContext.xml" />
                        </fileset>
                </copy>
                <copy overwrite="yes" todir="${build.classes.java}/${package}/web">
                        <fileset dir="${src.java.dir}/${package}/web">
                                <include name="*.properties" />
                        </fileset>
                </copy>
        </target>
```