Using the STPattern Plugin

Mahmoud A.Sakr, Ralf H.Güting

Database Systems for New Applications, FernUniversität in Hagen
58084 Hagen, Germany
`mahmoud.sakr@fernuni-hagen.de`
`rhg@fernuni-hagen.de`

August 14, 2009

1 Introduction

In this short manual, we describe how to install and use the STPattern Plugin. Before going on with this manual, we recommend you to get the Secondo *User Manual*, and the *SDK Installation Guide for your operating system* from the *Downloads/ Documentation* section in the Secondo web site `http://dna.fernuni-hagen.de/Secondo.html`, as we will refer to them in the sequel for more detailed information.

2 Installing the Secondo System

The Spatiotemporal Pattern Queries Plugin requires a running Secondo system of version 2.9.1 or later. This subsection shows how to install a Secondo system. More information can be found in the Secondo *User Manual*. You can skip this section if you already have a running compatible Secondo version.

The installation of Secondo is done in two steps:

1. Installing the SDK: The Secondo SDK contains all the third party tools required for compiling and running Secondo on your machine. Go to the Secondo web site at `http://dna.fernuni-hagen.de/Secondo.html`. Go to the *Downloads* page, section *Secondo Installation Kits*. Select the version for your platform (Mac-OS X, Linux, Windows). Get the installation guide and download the SDK. Follow the instructions to get an environment where you can compile and run Secondo. The SDK installation includes also a default Secondo version that will be copied to `$SECONDO_BUILD_DIR` which is set by default to `$HOME/secondo`.

2. Copying the Secondo source code: Go to the *Source Code* section of the *Downloads* page and download version 2.9.1. Extract it and replace the default Secondo version from the SDK installation by this version.

3 Installing the Spatiotemporal Pattern Queries Plugin

From the Secondo Plugins web site:
`http://dna.fernuni-hagen.de/Secondo.html/content_plugins.html`, get the files `Installer.jar`, `secinstall` [1] and the `STPatterns` Plugin. Inform yourself about the Secondo Plugins and follow the instructions on the web site to perform the installation. Then recompile Secondo (i.e. call `make` in directory `$SECONDO_BUILD_DIR`).

To make sure that the installation is successful, run Secondo (i.e. Navigate to `$SECONDO_BUILD_DIR/bin` and write `SecondoTTYNT`). At the Secondo

[1] If your java version conflicts with the Installer.jar file, you can generate the `Installer.jar` and the `secinstall` files by typing `make` in the `$SECONDO_BUILD_DIR/Tools/extensionInstaller` directory in the 2.9 version or later

prompt, write `list algebras`. Your installation is successful if you can find the `STPatternAlgebra` in the list.

4 Restoring a Database

After a fresh install of SECONDO, no databases are available on the system. We first restore the database *berlintest* that comes within the SECONDO distribution.

1. Navigate to the SECONDO bin directory `$SECONDO_BUILD_DIR/bin` and write `SecondoTTYNT`.
2. At the SECONDO prompt, write:

   ```
   restore database berlintest from berlintest
   close database
   quit
   ```

Please note that in `SecondoTTYNT`, you need to press carriage return twice after every command.

5 Experimenting with the Spatiotemporal Pattern Queries

In this section, we demonstrate how to run a pattern query and to visualize the results. We recommend you to read the SECONDO *User Manual* to know more details about using SECONDO modules.

5.1 Running the GUI

SECONDO has five user interfaces which are used for different use cases. To visualize the moving objects, one needs to run the *Javagui*. It is a window-oriented user interface that accepts queries and provides many viewers to visualize the results. To start the *Javagui*, first start SECONDO in the server mode. Navigate to the SECONDO bin directory `$SECONDO_BUILD_DIR/bin` and write

```
SecondoMonitor -s
```

To be able to write queries in an SQL-like syntax and invoke the SECONDO Optimizer, start the Optimization Server. In a new shell/bash, navigate to the SECONDO Optimizer directory `$SECONDO_BUILD_DIR/Optimizer` and write

```
StartOptServer
```

Now we launch the *Javagui*. In a new shell/bash, navigate to the SECONDO Javagui directory `$SECONDO_BUILD_DIR/Javagui` and write

```
sgui
```

The *Javagui* will now start and connect to the two running servers.

5.2 Viewing the Data

In the following examples, we use the database *berlintest* that is restored in section 4. The database includes many tables representing spatial and spatiotemporal data for the city of Berlin. In this subsection, we demonstrate briefly how to query and view parts of this data in the *Javagui*.

After starting the *Javagui* (as described in the previous subsection) make sure that the *Javagui* can connect to the SECONDO server and to the Optimizer server. You can do so by clicking on the *Server* and the *Optimizer* menus, and make sure that the *connect* and *enable* menu commands are disabled. Now open the database

```
open database berlintest
```

Query the spatial data for the underground train network.

```
query UBahn
```

In the "Representation" dialog that appears, you can specify the symbology for the retrieved data (ex: color, line width, ...). If you choose a symbology and wish to change it afterwards, choose the query from the *Object Manager* (the query list to the right) and click the *hide* button and then the *show* button.

Query the train stop *mehringdamm*

```
query mehringdamm
```

Query the snow storm *msnow*. It is a moving region that changes its position with time. To visualize moving objects, use the slider below the Command Panel.

```
query msnow
```

Query the moving trains. This query displays 562 trains that move on the underground network of Berlin. Use the slider to view the result.

```
query Trains
```

5.3 An Example Spatiotemporal Pattern Query

The following query looks for the trains which crossed through the snow storm *msnow*, and later in time passed by the stop *mehringdamm*. We will use this simple query to demonstrate how the spatiotemporal pattern queries in SECONDO work.

```
query Trains feed filter[ .  stpattern[pred1:  .Trip inside
msnow, pred2:  distance(.Trip, mehringdamm) < 10.0 ;
stconstraint("pred1", "pred2", vec("aabb"))] ] consume
```

where the `stpattern` is the spatiotemporal pattern predicate. It consists of two parts separated by a semicolon. The first part is a list of *lifted predicates* with *labels*. The second part is a list of `stconstraint` (stands for spatiotemporal constraint). An *stconstraint* specifies a temporal order between two predicates. This query is written in the SECONDO executable language as described in the User Manual. Run this query after the queries in the previous subsection and make the symbology of its results different from the query `query Trains` to best visualize.

In the *Javagui*, one can also write queries in an SQL-like language (described in the User Manual). The SQL-like query will be passed to the Optimizer server to generate an efficient executable plan. The previous query is now written SQL-like as follows

```
select * from trains where pattern([trip inside msnow as pred1,
distance(trip, mehringdamm) < 10.0 as pred2],
[stconstraint("pred1", "pred2", vec("aabb"))])
```

You can see the executable plan generated by the Optimizer in the *Object Manager*. It should look similar to the previous query. This is because the database does not contain indexes which can be used to optimize this query. Write the following command to create a spatial index on the units of the trip attribute of the trains relation.

```
optimizer createIndex(trains, trip, spatial(rtree, unit))
```

The SECONDO optimizer has a list of switches to control its behavior. We need to switch on some Optimizer options to enable our optimizer extensions to work.

```
optimizer setOption(improvedcosts)
optimizer setOption(rewriteInference)
optimizer setOption(rtreeIndexRules)
optimizer setOption(determinePredSig)
optimizer setOption(autoSamples)
optimizer setOption(autosave)
```

Now write again the query

```
select * from trains where pattern([trip inside msnow as pred1,
distance(trip, mehringdamm) < 10.0 as pred2],
[stconstraint("pred1", "pred2", vec("aabb"))])
```

Notice that the executable plan is changed so that it makes use of the newly created index.

If you want to experiment more with the plugin, try the following:

```
select * from trains where pattern([trip inside msnow as pred1,
distance(trip, mehringdamm) < 10.0 as pred2],
[stconstraint("pred1", "pred2", vec("bbaa"))])
```

Note that the pattern that we look for in this query is the reverse of the pattern in the query before. The query before has the simple temporal connector `vec("aabb")` which means `pred1` is fullfilled before `pred2`. This query uses `vec("bbaa")` which means the reverse of the query before.

```
let later = vec("aabb")
select * from trains where pattern([trip inside msnow as pred1,
distance(trip, mehringdamm) < 10.0 as pred2],
[stconstraint("pred1", "pred2", later)])

select * from trains where pattern([trip inside msnow as pred1,
distance(trip, mehringdamm) < 10.0 as pred2,
speed(trip) > 20.0 as pred3],
[stconstraint("pred1", "pred2", later),
stconstraint("pred2", "pred3", "later")])
```