


Note: Margin notes version!
Note: Spell check should be done!

Stairwalker

User Manual

We could write something here

DENNIS MULLER
JOCHEM ELSINGA

October 22, 2014
Universiteit Twente 

Contents

1	Introduction	4
2	Set Up	5
2.1	Database Setup	5
2.1.1	PostgreSQL Database Manager Setup	5
2.1.2	PostgreSQL Configuration	5
2.1.3	PostGIS Configuration in PostgreSQL	6
2.1.4	Serverside Stairwalker Extension in PostgreSQL	7
2.2	Pre-Aggregate Database Table	7
2.2.1	Description of Process	7
2.2.2	PreAggregate Tool	8
2.2.3	Current Status Support of Axis Types	9
2.2.4	Current Status Support of Aggregation Types	9
2.3	Tomcat Server Setup	10
2.4	Geoserver Deployment on Tomcat	10
2.4.1	Obtaining GeoServer and Aggregate Extension	10
2.4.2	Installing Extension	11
2.4.3	Deploying GeoServer	11
3	Deployment	12
3.1	Add Source	12
3.2	Setup Style	14
3.3	Adding Layer	15
3.4	Viewing and Using Layer	16
4	Development	18
4.1	Code Development	18
4.1.1	Creating Pre-Aggregate Index from Source	18
4.1.2	Filtering Words for Nominal Axis	19
4.2	Geoserver Visualization	20
4.2.1	Symbolizers	20
4.2.2	Filters	20
4.2.3	Additional Options	22
4.3	Client Side Development	24
5	Running Example	25
5.1	Requirements	25
5.2	Example Table Setup	26
5.2.1	Recreating Dataset	26
5.2.2	Creating Pre-Aggregate Index	26
5.3	GeoServer Setup	28
5.3.1	Add Source	28
5.3.2	Import Style	30
5.3.3	Create Layer	31

5.3.4	View Layer	32
-------	----------------------	----

1 Introduction

This intro still needs some work, priority is low.

This manual will explain how to use the Stairwalker Program of the Database faculty of the University of Twente. This manual will start by explaining on how to install all required components in order to have the basic running. This includes how to install all the required extensions for your database but also for other applications that are required to run this system. The second section will explain how to add all databases and different kind of datatypes to Geoserver, which is required to show in order to make the Stairwalker program run. Not only is this section going to explain how to show the data but it will give the option to customize this view and have it display data on all possibly ways. And the third section will explain how to make adjustments to the existing program and what future possibilities the system has. For example on how to add a new dimension to the aggregation tool or how to use a different aggregation type (median for example). The last section covers further development on the Stairwalker program in case something that hasn't been made yet needs to be added. This manual is made for a Postgres database other database programs are also possible, however this manual will not cover how to install certain extensions on those programs.

2 Set Up

In this section a detailed description will be given about setting up all the required peripheral programs to use stairwalker.

2.1 Database Setup

Currently the Stairwalker program only works with the PostgreSQL¹ and MonetDB² database management systems. This manual will only concern itself with PostgreSQL (often shortened to Postgres). Any further reference to a database implies a Postgres database.

In this section a walk through explaining the steps needed to setup PostgreSQL along with PostGIS³ and a custom made database extension for Stairwalker on an Ubuntu Linux operating system.

2.1.1 PostgreSQL Database Manager Setup

Installing Postgres on Linux should be straight forward. In the terminal search for PostgreSQL and then choose which version of PostgreSQL should be installed. Below the commands used to search and install version 9.1 of Postgres are shown.

1. `aptitude search postgresql`
2. `apt-get install postgresql-9.1`

For further information or if there are difficulties with the installation more details can be found on the installation page of the PostgreSQL website.

2.1.2 PostgreSQL Configuration

Once Postgres is installed on Linux two alterations will need to be made to the configuration files so that the database can be accessed from outside and by other users.

First off in the `Postgresql.conf` file the `listen_addresses` need to be changed from `localhost` to `all`. Assuming version 9.1 of Postgres this can be done as follows.

1. `cd /etc/postgresql/9.1/main`
2. `vi postgresql.conf`
3. *change* `listen_addresses = 'localhost'` to `listen_addresses = '*'`
4. *save and close*

¹PostgreSQL: <http://www.postgresql.org>

²MonteDB: <http://www.monetdb.org>

³PostGIS: <http://www.postgis.net>

Secondly in the file `pg_hba.conf` a line needs to be added to allow other users in Linux to access Postgres. This can be done as follows:

5. `vi pg_hba.conf`
6. `Add host all all 0.0.0.0/0 password`
7. *save and close*
8. `/etc/init.d/postgresql restart`

It should now be possible to create a database user and database in Postgres. More information about how this is done can be found on the Postgres manuals webpage.

2.1.3 PostGIS Configuration in PostgreSQL

The next step is to extend Postgres with PostGIS database expansion. This again should be straight forward, first search for PostGIS and then choose the version that goes with Postgres to install. Note in order to do the install root privileges are required. Below the commands to search and install PostGIS for Postgres-9.1 are shown.

1. `aptitude search postgis`
2. `apt-get install postgresql-9.1-postgis`

More information about installing PostGIS can be found on the PostGIS website.

Once PostGIS is installed some extra configuration is required. This needs to be done as postgres user. The commands are as follows.

Why do we do this again? What does it do?

1. `createlang plpgsql geonames`
2. `psql -f 'find/usr/share/postgresql/ -name postgis.sql -print' -d geonames`
3. `psql -f 'find/usr/share/postgresql/ -name spatial_ref_sys.sql -print' -d geonames`
4. `psql -f 'find/usr/share/postgresql/ -name postgis_comments.sql -print' -d geonames`

2.1.4 Serverside Stairwalker Extension in PostgreSQL

Finally there is C extension which allows processing of the SQL queries for the pre-aggregated database to happen on the server side instead of in the GeoServer application. This extension has to be installed on the Postgres database. The extension can be found in the directory `neogeo/pre-aggregate/src/db-extensions/postgres/pa_grid`.

The extension can be installed on Linux using the following commands.

1. *go to the pa_grid directory in the terminal*
2. `make`
this creates the dynamic library
3. `make install`
this installs the library in the postgres installation
4. `make sql`
declare the module in the database wanted

Note the extension has to be installed specifically for the database which will be used for pre-aggregation. In the makefile (also in the `pa_grid` folder) there is a `DATABASE` macro which should be set to the desired database.

Installing the extension also requires root access. Why was this again?

2.2 Pre-Aggregate Database Table

2.2.1 Description of Process

Stairwalker makes use of the Pre-Aggregate functions. These functions create a granularity in the dataset and calculate blocks at the lowest granularity for certain dimensions. Then subsequent blocks of higher granularity are built up from lower blocks. This creates a new dataset which summarizes the original dataset in terms of blocks which represent some significant data for every level of granularity and dimension.

I am not sure if this description is correct. Also I believe there is an eloquent way to describe the aggregation process.

Concretely take the dataset of tweets sent within the Netherlands. A pre-aggregation of this data could be in the following form. As significant data we consider the number of tweets in the x- and y-coordinate dimensions. We then set a highest granularity (zoom). The pre-aggregate algorithm then creates blocks bounded by the x- and y-coordinates at the highest granularity and for all those blocks calculates the number of tweets within those coordinates. Then each subsequent layer is built up from the previous layer. The final result is a new dataset which contains all blocks with the number of tweets in a coordinate block at each level of granularity.

2.2.2 PreAggregate Tool

To do the Pre-Aggregation step a tool has been developed which is explained here. The tool can be found in the directory `neogeo/pre-aggregate-tools/`. To generate the binary tools from source the Appassembler plugin of Maven is used. Run the following command to generate the tool.

```
mvn package appassembler:assemble
```

After successful completion of this command a new directory `appassembler` will have been created in the `target` directory containing a `repo` and a `bin` directory. The `bin` directory contains the actual binaries of the tool (in both Linux/Unix and Windows version) and the `repo` directory contains the tool dependencies. The tool should now be ready for use. An example of how to compile and use the tool to create a pre-aggregation index is given in Section 5.2.2.

The tool is used to create a Pre-Aggregate index for a table with n dimensions and a measure/aggregate column. It uses with the following commands:

```
usage: create-pa-index
  -axistosplit <axis index>    index of axis to split
  -chunksize <size>            maximum chunk size after split
  -config <file>               PreAggregate XML config file
  -d,--database <dbname>       name of database
  -dbtype <postgresql|monetdb> type of database
  -h,--host <host>             database host name or ip address
  -help                        prints this help message
  -p,--port <port>             port number of the database
  -password <password>         database password
  -s,--schema <schema>         schema name in the database
  -u,--user <user>             database username
  -v,--verbose                 Enable verbose output logging
```

The tool is dependent on the `PreAggregate.XML` config file which is used to define the PreAggregate index by specifying the column to aggregate, the type of aggregate that should be done and the dimensions to include. In the `neogeo/pre-aggregate-tools/` directory a sample configuration file is included.

How does this tool work nominal axis? More preprocessing may be required when first parsing words? Example `london_words` or `gender_words`

See Section 4.1 for more detail about creating a pre-aggregate index not using the special help tool.

Apart from creating a new pre-aggregate index table `<original-table-name>_pa` pre-aggregation also creates/updates two other tables, these are `pre_aggregate` and `pre_aggregate_axis`. These two tables are support tables for the aggregation. They contain information about which tables have been aggregated and which axis have been used in pre-aggregation.

2.2.3 Current Status Support of Axis Types

The axes with which a dataset will be split are referred to in the code as **AggregateAxes**. An **AggregateAxis** can be seen as a dimension in which the dataset will be divided. The term aggregate comes from the fact that these axis have a base size and all layers above the base size are built up by aggregating lower layer blocks. Currently the **AggregateAxis** can be split into two different subtypes. The first is a **MetricAxis** and the second is a **NominalAxis**. Each type will briefly be discussed below with examples of data types which can be split using each respective axis.

MetricAxis A **MetricAxis** can be seen as the default axis type, it supports a continues data type. Examples of such data types are coordinates and time.

Maybe this explanation should be extended with more details.

NominalAxis A **NominalAxis** is used to split non continues data. An example for such a data type is a word filter. A **NominalAxis** can be used to split the data depending on if a text contains x different words (predefined using a filter). In this case each previously created dimension is again split depending how many words in the filter the block contains.

Concretely if a word filter `{dog,cat,mouse,horse,NOFILTER}` is used and a data block represents some text with the sentence 'it's raining cats and dogs' then three splits would be made. One on the word `dog`, the second on `cat` and the third on `NOFILTER`.

2.2.4 Current Status Support of Aggregation Types

Currently there are 4 different aggregation types which can be used. These are discussed below. Note that there is an **ALL** option which returns all aggregate types in the pre-aggregate index. Furthermore these types can be used to create other types such as average.

The current options are as follows:

1. **ALL**: Returns all the aggregation types mentioned below.
2. **COUNT**: Returns the total amount of data in an aggregate box.
3. **SUM**: returns the total value of all data added up to each other. For instance a sum on the tweet length will result in the total amount of characters tweeted in a tile.
4. **MIN**: Returns the lowest value of all data that is aggregated. With the example of tweet length, it will returns the value of the lowest length tweet in that tile.
5. **MAX**: Returns the highest value of all data that is aggregated. With the example of tweet length, it will returns the value of the highest length tweet in that tile.

Small note, when using the pre-aggregate tool (created by Dennis) for the example ALL was used as it crashed when using COUNT, this should be looked into.

It is important to choose the right type in order to get a good representation of the data. To show the total amount in a tile, one should chose COUNT as this will returns the total number of data points in a tile. Whereas if one wants only the highest value in the data (for example the highest building in an area) then MAX should be used.

2.3 Tomcat Server Setup

To visually display the aggregated dataset a third party application is used. This application is called GeoServer, GeoServer is an open source server which can be used to display geospatial data. In order to run GeoServer a web server is needed.

During development the Apache Tomcat⁴ was used for this purpose. The use of Tomcat is not required for Stairwalker, alternative options may also be used as long as it is possible to deploy WAR files. Tomcat was used during development therefore a description of how to obtain Tomcat is given here. Furthermore whenever web hosting is mentioned in this manual it will be assumed Tomcat is being used.

Information about setting up a Tomcat server can be found on the Apache Tomcat website.

2.4 Geoserver Deployment on Tomcat

2.4.1 Obtaining GeoServer and Aggregate Extension

GeoServer⁵ is used to visually display the collected data, this is done using the pre-aggregate index. To use the aggregated dataset an extension has be written for GeoServer which needs to be included in the web application. This is done by including the JAR files of the extension in the WEB-INF of the GeoServer WAR file. A step by step guild is given below in Section 2.4.2. However before following these instructions the necessary JAR and WAR files will need to be obtained.

The GeoServer WAR file can be downloaded from the GeoServer website. The custom Java extension files should be built using the source code. The extension consists of the **pre-aggregate** and **geoserver-ext** projects in the **neogeo** project. Note that first the JAR file of **pre-aggregate** should be created as it is a dependency of **geoserver-ext**. The JARs can be built using the command line or in a IDE such as Eclipse or Netbeans, using the **clean** and **build** command on a project in Netbeans should build **<project-name>-0.0.1-SNAPSHOT.jar** which can be found in the **target** directory of the respective project.

⁴<http://tomcat.apache.org/>

⁵GeoServer: <http://www.geoserver.org>

2.4.2 Installing Extension

These next instructions assume the geoserver extension files are located in the directory `/data/upload/` and that the `geoserver.war` file is located in the directory `/data/tmp/tmp_war`. If this is not the case the file paths in the instructions below should be changed accordingly.

Unpack the WAR file

1. `jar -xvf geoserver.war`

Copy the JAR files of the extension into `WEB-INF/lib` directory of the GeoServer unpacked WAR file

2. `cp /data/upload/pre-aggregate-0.0.1-SNAPSHOT.jar /data/tmp/tmp_war/WEB-INF/lib`
3. `cp /data/upload/geoserver-ext-0.0.1-SNAPSHOT.jar /data/tmp/tmp_war/WEB-INF/lib`

Recreate the GeoServer WAR file

4. `jar -cvf geoserver.war META-INF/ WEB-INF/ index.html data/`

The GeoServer WAR with the stairwalker extension should now be ready for deployment.

2.4.3 Deploying GeoServer

After the GeoServer WAR file has been repacked with the aggregate extension included it is ready to be deployed in a web server. Section 2.3 describes how to set up such a Tomcat web server. Once the server is running it can be accessed locally with `http://localhost:8080/` assuming default installation configuration were used, otherwise the port number might be different. In the case that the Tomcat server is installed on a difference machine then the web server can be accessed by replacing `localhost` with the IP address of that machine.

From the Tomcat homepage it should be possible to access the Tomcat manager webapp. With the default setup it should be possible to login with the following credentials:

username: manager

password: tcmanager

In the Tomcat manager webapp under the `deploy` section it is possible to upload a WAR file to be deployed. Select the repacked WAR file from Section 2.4.2 and deploy the application. Once the application is deployed it will be displayed in the `application` section of the Tomcat manager webapp. From there it is possible to follow the path given for the GeoServer application or, if the default configuration was used to go to `http://<Tomcat-IPaddress>:8080/geoserver/web/`.

3 Deployment

This section will give a detailed description of how to import an aggregated database table into GeoServer to get a visual representation of the dataset. First instructions will be given on how to link the database table to GeoServer. Next creating styles and layers for data representation will be discussed. The final sub section will discuss how to view the data using GeoServer. For this section it is assumed that all the initial preparation discussed in Section 2 has been completed.

GeoServer should already be deployed on a web server (see Section 2.4.3), and can then be accessed with `http://<Tomcat-IPaddress>:8080/geoserver/web/`. It is required to login in to the GeoServer web administration interface. When using the default setup of GeoServer the login credentials are:

username: admin

password: geoserver

A concrete example of how to fully deploy a pre-aggregated index can be found in Section 5.3.

3.1 Add Source

Once logged in to the web administration interface it is possible to add a new data store to GeoServer. Instructions of how to add a new **NeoGeo Aggregate** vector data source which contains the aggregate index created in Section 2.2.2 to the stores in GeoServer.

1. Navigate to **Stores** by clicking on **Stores** link under the **Data** section in the navigator on the left hand side of the web administration interface homepage.
2. On the **Stores** page select the option **Add new Store** located at the top of the page. This leads to a page titled **New Store chooser**.
3. In the list of **Vector Data Sources** the option **NeoGeo Aggregate** should be present, choose this format for the data source.

If the option **NeoGeo Aggregate** is not available it means the GeoServer extension from Section 2.4.2 was not done correctly.

4. Clicking **NeoGeo Aggregate** will open a new page titled **New Vector Data Source** in which several fields have to be filled out, explanation of mandatory fields can be found in the list below on page 13.
5. For an express setup the fields which have already been filled can remain the same.
6. Once all the required fields are filled in click the **Save** button.
7. A new **NeoGeo Aggregate** source is now created and can be view and edited in **Sources**.

8. After saving GeoServer opens the page **New Layer** on which new layers can be created using the **Data Source**. How this is done is discussed in Section 3.3.

Here a list of all mandatory fields on the **New Vector Data Source** page with explanation.

- **Data Source Name** - An arbitrary name which will be assigned to the store.
- **Database type** - The type of underlying database, either PostgreSQL or MonetDB.
- **Hostname** - Hostname of the database server where the aggregation index is maintained.
- **Port** - Port number of the database.
- **Schema** - Name of the schema where the aggregation index is maintained.
- **database** - Name of the database where the aggregation index is maintained.
- **Username** - Username of the used database.
- **Password** - Password of the used database.
- **xSize, ySize, timeSize** - Specifies the dimensions of the grid which is created for every view of the map to calculate the aggregates per cell. The higher the number of cells the more detailed the information.

Are the xSize, ySize, timeSize dimensions not already set in the source?
What happens when timeSize > 1?

- **count, sum, minimum, maximum** - Select the boxes of the aggregates which will be used in the visualization. Note that these basic aggregates more aggregates such as mean can be derived.
- **Enable server-side Stairwalker** - Selecting will cause the data source to rely on the use of the database plugins to use the Pre-Aggregate index. For performance reasons it is highly recommended to use this option. See Section 2.1.4 for more details.

The Enable server-side Stairwalker option is no longer available? Remove?

- **Enable query logging** - Selecting will turn on the logging of all Pre-Aggregate queries into a separate table called **pre_aggregate_logging**.

3.2 Setup Style

In GeoServer styles are used to render, or make available, geospatial data. Styles are used to visually represent the aggregation index which is represented in a layer. In GeoServer layers are written in Styled Layer Descriptor (SLD) which is a subset of XML. GeoServer comes setup with several different styles however to get the most out of the dataset it is best to develop a style specific to the layer which represents that data.

In this section only instructions on how to add new styles to GeoServer are given. For information on how to edit styles see Section 4.2 or the GeoServer user manual⁶ which gives an in depth guide on developing styles.

1. Navigate to **Styles** by clicking on the **Styles** link under the **Data** section in the navigator on the left hand side of the web administration interface homepage.
2. On the **Styles** page select the option **Add a new style** located at the top of the page.
3. A new page titled **New Style** should open. There are now two possibilities, either a new style can be developed completely in the browser or a SLD file can be imported.
4. To import an already created SLD file scroll to the bottom of the page and press the **Choose File** button.
5. Select the style which should be imported and then press **Upload...** in the browser.
6. This fills in the **Name** field with the name of the file and the SLD editor with the content of the file.
7. It is possible to check the syntax of the SLD code by pressing the **Validate** button at the bottom of the page. At the top the page GeoServer will give feedback on the SLD code, either error messages or a no validation errors message.
8. Finally the style can be saved by pressing the **Submit** button at the bottom of the page.

We should maybe mention something about the workspace? What does **nurc** represent? It is even important to choose a workspace?

⁶<http://docs.geoserver.org/stable/en/user/styling/index.html#styling>

3.3 Adding Layer

In GeoServer a layer refers to raster or vector data that contains geographic features. Layers represent each feature (axis in the pre-aggregate index) of a dataset that needs to be represented. All layers much have a source of the data which in this case was setup in Section 3.1. More information about layers can be found in the GeoServer User Manual.

Creating a layer for a pre-aggregate index dataset can be done as follows:

1. Navigate to **Layers** by clicking on the **Layers** link under the **Data** section in the navigator on the left hand side of the web administration interface homepage.
2. On the **Layers** page select the option **Add a new resource** located at the top of the page.
3. This leads to a new page where the **Store** which contains the layer needs be chosen from a drop-down list. If there are no **Stores** available make sure one was added, see Section 3.1
4. Choose the **Store** in which the aggregation index is stored.
5. Once a **Store** is selected a list of resources contained in the **Store** is given. These resources are the different aggregated indexes in the database which was linked to a **Store** in Section 3.1.
6. Select the pre-aggregate index which should be visualized in a layer by clicking the **Publish** link corresponding to the **Layer name** of the aggregate index.

At this point a layer has been selected to be published. This layer will be a visual representation of the data from the aggregation index created in Section 2.2.2. In order to make sure the correct geographical location is used in GeoServer and to give the layer a fitting style the following steps have to be taken in on the **Edit Layer** page.

7. In the **Data** tab the following sections and fields should be filled in.
 - (a) In the **Basic Resource Info** there are some labeling fields. Standard the **Name** and **Title** are `<aggregation-index-tablename>` followed by `___myAggregate`. These can both changed to whatever is desired. However make sure the that the **Enabled** box is ticked in this section.
 - (b) In the **Coordinate Reference Systems** section there are three fields.
 - i. **Native SRS** should be `EPSG:4326`.
 - ii. **Declared SRS** should be `ESPG:3857`. This coordinate system is used since it is what is usually used for tile based map representation.

- iii. **SRS handling** should be **Reproject native** to **declared**.
 - (c) In the **Bounding Boxes** the coordinates corresponding to the data from the aggregation index is calculated for GeoServer. For **Native Bounding Box** click **Compute from data** and for **Lat/Lon Bounding Box** click **Compute from native bounds**.
 - (d) All other sections in this tab are of little importance in a basic deployment.
8. Next in the **Publishing** tab a style can be added to the layer, the default style of a layer is **polygon**. In the section **WMS Settings** the field **Default Style** can be changed by selecting the desired style from the drop-down menu. For more about styles and creating styles see Section 3.2 and Section 4.2.

If the aggregation index does not contain a time dimension the setup of the layer is now complete and can be saved. However if the aggregation index does have a time dimension some additional adjustments need to be made which are described below.

9. Select the **Dimensions** tab.
- (a) Enable the the **Time** dimension.
 - (b) As **Attribute** select **starttime**.
 - (c) Do not set an **End Attribute**.
 - (d) As **Presentation** select **Continuous interval**.
10. Save the layer.

The layer which represents the dataset with a style created in Section 3.2 has now been created and is ready for use. A layer can be edited once it has been created so if changes need to be made a new layer should not be created. Section 3.4 shows how to preview a layer and Section 4.3 discusses how to use a GeoServer layer with OpenLayers⁷ to a geospatial visualization of the data on a web page.

3.4 Viewing and Using Layer

In GeoServer it is possible to get a preview of layer such as the one created in Section 3.3. Previewing a layer can be done as follows:

1. Navigate to **Layer Preview** by clicking **Layer Preview** link under the **Data** section in the navigator on the left hand side of the web administration interface homepage.
2. The **Layer Preview** page will have a list of all configured layers with can be previewed in various formats.

⁷<http://openlayers.org/>

3. Locate the layer which should be shown and from the `All Formats` column choice any `WMS` format.
4. After selecting a format to view the layer a new page will open with a visual representation of the top most layer of dataset.

Note that other preview formats should also be possible. For example it is possible to use the OpenLayers preview format which allows one to navigate the geospatial data. In section Section 4.3 OpenLayers will also be used to visualize the dataset on a web page. One drawback is that for every movement in the preview a new query has to be calculated. When server side stairwalker (Section 2.1.4) is not setup calculating can be time consuming. Therefore during development of the pre-aggregation index and testing it is advices to use a static preview and only once everything works as desired to use a dynamic preview.

If the layer contains a nominal axis it is possible to alter the value of the nominal with which the data is filtered. This is done by adding a parameter in the request made to the GeoServer extension. By extending the `HTML` request sent to GeoServer with `&VIEWPARAMS=<TYPE>:<VALUE>;` the nominal axis filter is used. Currently the extension only supports the nominal type `keyword`. If the pre-aggregate index was created using a nominal axis (splitting on `VALUES`) then using `&VIEWPARAMS` will split the visualization on a give `VALUE`. `&VIEWPARAMS` can accept more `<TYPE>:<VALUE>;` tuples, however parsing type will need to be extended on in the code. For more information see Section 4.1.2.

It may be that a preview fails to load, this can be due to two reasons. The first is an error in the style, in this case the style needs to be tested which can be done by validating the style in GeoServer. For more information about styles see Section 4.2. The second reason is an error is creating an `SQL` query for the pre-aggregated index. If no mistakes where made during setup and in creating a pre-aggregate index there will be a need to dive into the code where detailed logging is done.

This would be a good point to reference the logging done in the code however i cannot find where this log file is located, debugging is also possible using the console in an IDE.

4 Development

4.1 Code Development

In this section some pointers will be given to important sections of code in terms of processing pre-aggregate indexes which are used by the GeoServer extension.

During the development of Stairwalker, the developers project of GeoServer, the source code used is available to download on the GeoServer website. Using the source code and running GeoServer from an IDE provides useful when troubleshooting, as logging information is printed directly to the console, and for constant redeployment a new WAR file does not have to be remade every time. The source code of the Stairwalker project is available on GitHub⁸. The two important packages from this project are `geoserver-ext` and `pre-aggregate`. The first is the extension used in GeoServer to handle pre-aggregation indexes as data sources and the second is the package which is used to create the pre-aggregate index.

4.1.1 Creating Pre-Aggregate Index from Source

This subsection is still superficial in some aspects. I am unsure how extensive an explanation is required here and if it is even necessary.

Creating a pre-aggregate index can be done using a tool as shown in Section 2.2.2 however if this fails to work or some unimplemented functionality is required it is also possible to create a new method in the `Test.java` of `pre-aggregate` in which the steps for creating the pre-aggregation index can be done manually. In the `main` of this class configuration file `database.properties` is read, which contains login information about the database which contains the dataset, and connection is made to said database. Next a pre-aggregate index is made using the custom made method which describes the pre-aggregation. How to set up such a method is described in this subsection.

Firstly for each axis on which the dataset will be split an `AggregateAxis` variable will be defined. Next these variables should be initiated with a axis type (`metric` or `nominal`), in the case of `nominal` also a word list (which contains the words on which the dataset will be split) should be created. The constructors of both axis types are as follows:

```
public MetricAxis(String columnExpression,      1
                  String type, Object BASEBLOCKSIZE, short N);  2
                                                                    3
public NominalAxis(String word_collection_column,  4
                   String word_index_column, String wordlist_str,  5
                   String name);                  6
```

Once all axes have been initiated an array containing all axes should be created which will be used as input for the `PreAggregate` variable created in the

⁸<https://github.com/utwente-db/neogeo>

next. The next step is to create a pre-aggregate index by creating and initiating a `PreAggregate` variable. The constructor of `PreAggregate` is as follows:

```
public PreAggregate(Connection c, String schema,      1
                    String table, String override_name, 2
                    String label, AggregateAxis axis[], 3
                    String aggregateColumn, String aggregateType, 4
                    int aggregateMask, int axisToSplit, 5
                    long chunkSize, Object[][] newRange); 6
```

Afterwards the connection to the database should be closed. A method containing these components is created it can be statically run in the main. Note that with the `NominalAxis` some preprocessing maybe required on the dataset. In order to do this help method `tagWordIds2Table` is used. On the `NominalAxis` created all the `tagWordIds2Table` method which has the following arguments:

```
public void tagWordIds2Table(Connection c,      1
                             String schema, String org_table, 2
                             String new_table); 3
```

4.1.2 Filtering Words for Nominal Axis

In the package `geoserver-ext` specifically the class `AggregationFeatureSource` the SQL query which requests data from the pre-aggregated index for GeoServer is built. If the pre-aggregated index contains a `Nominal` axis then it is possible to pass along words to filter the axis on in GeoServer using `VIEWPARAMS` (see Section 3.4). In order to include this filter parameter the method `getReaderInternal` and possibly `reformulateQuery` should be extended.

In the method `getReaderInternal` the layer request sent to GeoServer is parsed including the `VIEWPARAMS`, so a variable should be created in the method which matches the word which should be filtered given by `<TYPE>:<VALUE>`. This variable can then be passed to the `reformulateQuery` method. It is then possible in `reformulateQuery` for a specific `AggregateAxis` to split on the variable passed along in `getReaderInternal`.

4.2 Geoserver Visualization

This section is going to discuss most of the possibilities that GeoServer has to offer when it comes to the visualization of the data. The GeoServer manual has some information on this subject, which can be found on: <http://docs.geoserver.org/2.5.x/en/user/styling/sld-reference/index.html>.

The following sections will give some more information on what the differences are between the different options and some ways to implement the different options. This section will use some of the examples used in our demo to give some more insight what can be done and hopefully making it easier to realize what is wanted. Most information can be found on the website, below will be a discussion on what types are best used when, but also some more information on how to make them work properly.

4.2.1 Symbolizers

In SLD there are three different symbolizers, a linesymbolizer, a pointsymbolizer and a polygonsymbolizer. A pointsymbolizer is used when the data that has to be represented is best shown as points. It does exactly what it says, you'll get a map with points on it and each point will represent a data-object from your dataset. This symbolizer can be really handy in certain situations. For example when you want to show all locations where a rare species of a bird has been found it will show a map with all points where a bird has been reported.

A linesymbolizer is used when the data that has to be displayed is best shown in lines. This symbolizer is best used to display roads for example. It isn't a symbolizer that can be used to represent data very well, but it is more used for pre-defined data, think of rivers, roads etc.

A polygonsymbolizer is used when the data that you want to represent has to be displayed in two-dimensional objects. There are many possibilities in a polygonsymbolizer. It is possible to make a simple square but it also has the option to make circles or triangles. It is one of the most commonly used symbolizers. It has the most potential since you can do a lot with a polygonsymbolizer. A good example where a polygonsymbolizer is used, is to display the amount of people living in cities. This can be done with a circle polygon and that the circle will get bigger when more people live in a city.

4.2.2 Filters

Filters are the most important function when it comes to making a custom style. A filter is basically the basis of a fancy layer. What a filter does is that it makes a ruling and if that ruling is met, the color, labeling etc will be done. In SLD it is possible to have an unlimited amount of filters so the possibilities are endless. The following filter expression can be used:

- `<PropertyIsEqualTo>`
- `<PropertyIsNotEqualTo>`

- <PropertyIsLessThan>
- <PropertyIsLessThanOrEqualTo>
- <PropertyIsGreaterThan>
- <PropertyIsGreaterThanOrEqualTo>

An example on how a single filter can be used is the following:

```
<ogc:Filter>
  <ogc:PropertyIsLessThan>
    <ogc:PropertyName>testvalue</ogc:PropertyName>
    <ogc:Literal>200</ogc:Literal>
  </ogc:PropertyIsLessThan>
</ogc:Filter>
```

This example will test if the `testvalue` is less than 200. If this is the case you can add what the filter should be doing. Below is the complete example that does something with this filters.

```
<Rule>
  <Name>SmallPop</Name>
  <Title>Less Than 100</Title>
  <ogc:Filter>
    <ogc:PropertyIsLessThan>
      <ogc:PropertyName>testvalue</ogc:PropertyName>
      <ogc:Literal>100</ogc:Literal>
    </ogc:PropertyIsLessThan>
  </ogc:Filter>
  <PolygonSymbolizer>
    <Fill>
      <CssParameter name="fill">#38FF19
    </CssParameter>
      <CssParameter name="fill-opacity">1.0
    </CssParameter>
    </Fill>
    <Stroke>
      <CssParameter name="stroke">#000000
    </CssParameter>
      <CssParameter name="stroke-width">1.0
    </CssParameter>
    </Stroke>
  </PolygonSymbolizer>
</Rule>
```

What this example does is that if the `testvalue` is below 100, it will fill a polygon with the color: `#38FF19`. If this is not the case it will go to the next rule (if there is any otherwise it will just not do anything). The image shows

a graph of the implementation we made for our data. The image has different kind of colors for the amount of data in a tile. If the amount is high the color will become more red and if there is little data the tile will be green. This is a good example on what can be done with filters.

4.2.3 Additional Options

GeoServer SLD has a lot of options when it comes to customizing the data display that you've made. Below are some of the important features that are commonly used in GeoServer.

Halo: A halo gives a glow behind the current label. It should always be used in a textsymbolizer, since this is the only place you can add a halo. To use a halo its very simple, you do `<Halo> </Halo>` and in between it is possible to add `<Radius>` and `<Fill>`. For more information on how to use a `<Fill>` look in the GeoServer SLD cookbook.

Anchorpoint: An anchor point is a really handy tool to place your label on every place possible. It is used as shown below and important to notice is that you can set where the anchor point is (for example above the point) and you can displace it afterwards based on this anchor point, for instance make it go all the way to left (negative X placement) or all the way to the right (positive X placement)

```
<PointPlacement>
  <AnchorPoint>
    <AnchorPointX>0.5</AnchorPointX>
    <AnchorPointY>0.0</AnchorPointY>
  </AnchorPoint>
  <Displacement>
    <DisplacementX>0</DisplacementX>
    <DisplacementY>25</DisplacementY>
  </Displacement>
  <Rotation>-45</Rotation>
</PointPlacement>
```

Opacity: Opacity is the transparency of either a label, point, polygon or line. It can be used to paint layers over each other (setting Opacity to 0), this is something used a lot in case multiple data has to be displayed in the same tile (used in our example as well). The way you use it is the following:

```
<Opacity>0.3</Opacity>
```

Rotation: Rotation is the function that is used to turn all shapes and labels in SLD. It is very handy if you want to turn tiles or make labels line up with lines better. The way to use it is very simple in the section that has to be rotated just add the following code: `<Rotation>-45</Rotation>` for a negative 45 degree turn.

Graphic Fill: A graphic fill is used in case a picture/image has to be shown in a layer. It has a lot of possibilities since every picture/image can be added

through this way. The implementation is a little more complex so below is an example of a graphic fill.

```
<FeatureTypeStyle> 1
  <Rule> 2
    <PolygonSymbolizer> 3
      <Fill> 4
        <GraphicFill> 5
          <Graphic> 6
            <ExternalGraphic> 7
              <OnlineResource> 8
                xlink:type="simple" 9
                xlink:href="colorblocks.png" /> 10
              <Format>image/png</Format> 11
            </ExternalGraphic> 12
            <Size>93</Size> 13
          </Graphic> 14
        </GraphicFill> 15
      </Fill> 16
    </PolygonSymbolizer> 17
  </Rule> 18
</FeatureTypeStyle> 19
```

There are a lot more options and a lot of the information can be found in the SLD cookbook on the GeoServer website. This section was meant to give some more insight the commonly used functions.

4.3 Client Side Development

Here Mathijs will write a subsection on what is possible in terms of client side development with respect to the stairwalker project.

5 Running Example

In this section a concrete example is given to show how use Stairwalker. The example runs from creating a pre-aggregation index from a given dataset to geospatially representing the data using GeoServer. For the example a dataset is provide in terms of a .csv file. This is an exported database table which contains tweets sent in England.

The result expected in this example can be seen in Figure 9. Different colors represent a the number of tweets in a region. The result only shows the highest granularity, it is clear to see in the middle of the image that the most tweets are sent. If this layer was is combined with a map it would be clear that this region is directly above England

5.1 Requirements

Before recreating the given example all necessary installations should be made and the required files obtained. The following programs should be installed:

1. PostgreSQL (Section 2.1.1)
2. PosgGIS for a PostgreSQL database (Section 2.1.3)
3. Tomcat or another web service server (Section 2.3)
4. GeoServer with extension (Section 2.4.3)

Also the following files should be at the ready:

1. The tool to make a pre-aggregate index: Pre-Aggregate-Index tool (Section 2.2.2)
2. Configuration file for the tool: `runningexample.config.xml`
3. Sample dataset: `RunningExample.csv`
4. Sample SLD file: `RunningExemplSLD.xml`

Currently the files can be found in the `RunningExample` directory in the same Git as this manual.

Furthermore it is useful to have a tool in which the PostgreSQL database can be managed. During development the tool pgAdmin⁹ was used.

⁹<http://www.pgadmin.org/>

5.2 Example Table Setup

The first step of the process is to have a dataset which will be aggregated. For this example a dataset is supplied in the form of a `.csv` file. In order to import this first recreate the table in the database. This can be done with the following SQL query:

5.2.1 Recreating Dataset

```
CREATE TABLE runningexample (  
    id_str character varying(25),  
    tweet text,  
    user_name text,  
    place_name text,  
    time timestamp with time zone,  
    reply_to text,  
    place_id bigint,  
    len bigint,  
    coordinates geometry,  
    CONSTRAINT enforce_dims_coordinates CHECK ((  
        st_ndims(coordinates) = 2)),  
    CONSTRAINT enforce_geotype_coordinates CHECK (  
        (((geometrytype(coordinates) = 'POINT'::  
            text) OR (coordinates IS NULL))),  
    CONSTRAINT enforce_srid_coordinates CHECK ((  
        st_srid(coordinates) = 4326))  
);
```

Once the table is created import the `.csv` file to the table.

Note that `RunningExample.csv` contains column headers, uses `;` as column separators and `"` as quote separators. After the import is done a pre-aggregate index can be made.

5.2.2 Creating Pre-Aggregate Index

An in depth discussion of the use of the pre-aggregate index tool is given in Section 2.2.2. In the example only commands will be given with little explanation.

First the tool needs to be compiled which can be done with the command below executed in the `pre-aggregate-tools` directory.

```
mvn package appassembler:assemble
```

The next step is to put the pre-aggregate tool config file in the `pre-aggregate-tools` directory. Once this is done the tool can be called with the following command. Note some variables will be different than in the listing below. These are `<database>`, `<host>`, `<port>`, `<pass>`, `<user>`, these should be filled out according how PostgreSQL was set up.

```
target\appassembler\bin\create-pa-index -config  
runningexample.config.xml -d <database> -dbtype
```

```
postgresql -h <host> -p <port> -password <pass> -s  
public -u <user>
```

This will create a pre-aggregate index of the dataset. In the database there will be three new tables. The pre-aggregate index named **runningexample_pa** and two help tables which keep track of the indexes and the axes used by those indexes. All the work on the side of the database is now done and the next step is to visualize the dataset using GeoServer.

5.3 GeoServer Setup

This section will give a step by step guide of how to create a visual geospatial representation using the pre-aggregated index of the example dataset. This will be done using GeoServer, specifically the GeoServer web administration interface. This section offers a concrete version of the deployment discussed in Section 3.

Figure 1 shows the **Data** section of the navigator which can be found on the left hand side of the web administration interface. The links in this section will be used to navigate between different pages needed to configure the whole setup.

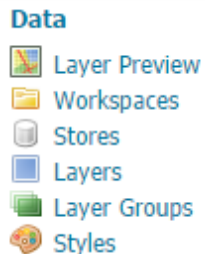
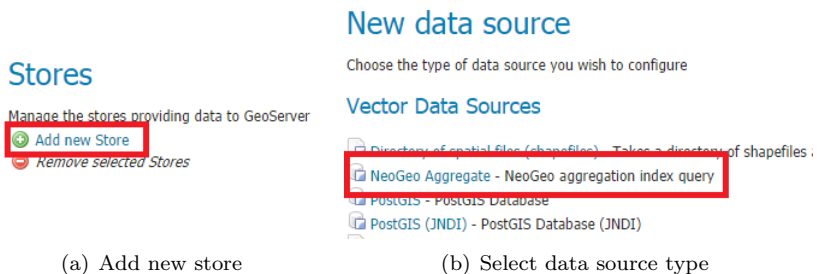


Figure 1: Data section of navigator

5.3.1 Add Source

A data source is added in the following manor:

1. Click on the **Stores** link in the **Data** section shown in Figure 1.
2. The **Stores** will open, the top of the page will look like Figure 2(a), click on the **Add new Store** link.
3. A selection of different **Vector Data Sources** is now available. Select **NeoGeo Aggregate** as shown in Figure 2(b).



(a) Add new store

(b) Select data source type

Figure 2: Adding new **Store** to GeoServer

4. After selecting **NeoGeo Aggregate** as **Vector Data Source** a page like Figure 3 will open. Fill in all the fields as shown, some values may differ depending on how the database is setup. More exact information can be found in Section 3.1.
5. Once everything is filled out click the **Save** button. This leads to page where **Layers** can be published. However before that is done first the **Style** should be imported.

New Vector Data Source

Add a new vector data source

NeoGeo Aggregate
NeoGeo aggregation index query

Basic Store Info

Workspace *

nurc

Data Source Name *

RunningExampleDataStore

Description

☒ Enabled

Connection Parameters

Database type *

POSTGRES

hostname *

localhost

port *

5432

schema *

public

database *

TestDatabase

Username *

postgres

Password *

Namespace *

http://localhost/nurc

xSize *

10

ySize *

10

timeSize *

1

☒ count

☐ sum

☐ minimum

☐ maximum

☐ enable query logging

Save

Cancel

Edit Layer

Edit layer data and publishing

nurc:RunningExampleLayer

Configure the resource and publishing information for the current layer

Data Publishing Dimensions Tile Caching

Basic Resource Info

Name

RunningExampleLayer

☒ Enabled

☒ Advertised

Title

RunningExampleLayer

Abstract

Keywords

Current Keywords

features
aggregate_runningexample_myAggregate

Remove selected

New Keyword

Vocabulary

Add Keyword

Metadata links

No metadata links so far

Add link

Note only FGDC and TC211 metadata links show up in WMS 1.1.1 capabilities

Coordinate Reference Systems

Native SRS

EPSG:4326

EPSG:WGS 84...

Declared SRS

EPSG:3857

Find...

EPSG:WGS 84 / Pseudo-Mercator...

SRS handling

Reproject native to declared

Bounding Boxes

Native Bounding Box

Min X

Min Y

Max X

Max Y

-0.12

51.327

0.449

51.658

Compute from data

Lat/Lon Bounding Box

Min X

Min Y

Max X

Max Y

-0.000010779783

0.0004610782858

0.0000040334356

0.0004640517094

compute from native bounds

Feature Type Details

Property

Type

countaggr

Long

area

Polygon

Reload feature type

Save

Cancel

Figure 3: New Vector Data Source

Figure 4: Edit Layer page

5.3.2 Import Style

Importing a style is done as follows:

6. Click on the **Styles** link in the **Data** section shown in Figure 1.
7. Click on the **Add a new style** button which will go a page similar to Figure 5 although empty.
8. Import the **RunningExampleSLD.xml** file by using the **Choose File** button then the **Upload...** link highlighted in red in Figure 5.
9. Once the style has been upload the **New style** page should look like Figure 5.
10. Press the **Save** button.

The style used in this example has been imported in GeoServer and now the layer is ready to published.

New style

Type a new SLD definition, or use an existing one as a template, or upload a ready made style from your file system. The editor can provide a valid SLD document.

Name

Workspace

Copy from existing style

Choose One

12pt

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <StyledLayerDescriptor version="1.0.0"
3 xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
4 xmlns="http://www.opengis.net/sld"
5 xmlns:ogc="http://www.opengis.net/ogc"
6 xmlns:xlink="http://www.w3.org/1999/xlink"
7 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
8 <!-- a Named Layer is the basic building block of an SLD document -->
9 <NamedLayer>
10 <Name>polygon_nt_color1</Name>
11 <UserStyle>
12 <!-- Styles can have names, titles and abstracts -->
13 <Title>Default Polygon with color based on cnt 1</Title>
14 <Abstract>A sample style that draws a polygon and adds the feautre as text</Abstract>
15 <!-- FeatureTypeStyles describe how to render different features -->
16 <!-- A FeatureTypeStyle for rendering polygons -->
17 <FeatureTypeStyle>
18 <Rule>
19 <Name>LabelPOP</Name>
20 <Title>Less Than 1000000</Title>
21 <ogc:Filter>
22 <ogc:PropertyIsLessThan>
23 <ogc:PropertyName>countaggr</ogc:PropertyName>
24 <ogc:Literal>1000000</ogc:Literal>
25 </ogc:PropertyIsLessThan>
26 </ogc:Filter>
27 <TextSymbolizer>
28 <Label>
29 <ogc:PropertyName>countaggr</ogc:PropertyName>
30 </Label>
31 <Font>
32 <CssParameter name="font-family">Times</CssParameter>
33
```

SLD file

Choose File

Figure 5: Importing SLD style from file

5.3.3 Create Layer

New Layer

Add a new layer

You can create a new feature type by manually configuring the attribute names and types. [Create new feature type...](#)
Here is a list of resources contained in the store 'RunningExampleDataStore'. Click on the layer you wish to configure

<< < > >> Results 1 to 1 (out of 1 items)		Search	
Published	Layer name	Action	
	aggregate_runningexample___myAggregate	Publish	
<< < > >> Results 1 to 1 (out of 1 items)			

Figure 6: Publishing a Layer

Creating a new layer is done as follows:

11. Click on the **Layers** link in the **Data** section shown in Figure 1.
12. This opens the **Layers** page, here click on the **Add a new resource** button. This opens a page similar to Figure 6.
13. Select the **Publish** action for the example layer.
14. A page like Figure 4 will open. Set highlighted fields to match Figure 4. More exact information about these fields can be found in Section 3.3.
15. After the fields in the **Data** are filled in, go to the **Publishing** tab, see Figure 7.
16. Set the default style to **RunningExampleSLD** like in Figure 7.
17. Press the **Save** button.

Edit Layer

Edit layer data and publishing

nurc:RunningExampleLayer

Configure the resource and publishing information for the current layer

Data	Publishing	Dimensions	Tile Caching
------	------------	------------	--------------

HTTP Settings

☐ Response Cache Headers

Cache Time (seconds)

WMS Settings

☒ Queryable

☐ Opaque

Default Style

RunningExampleSLD

Less Than 10000000

☒ Less Than 1

☐ Less Than 6

☐ Less Than 100

☐ Less Than 500

☐ 100 to 250

☐ 250 to 2500

☐ 2500+

Figure 7: Adding a Style to the Layer

A layer for the example dataset has now been created and is ready to be viewed.

5.3.4 View Layer

Layer Preview

List of all layers configured in GeoServer and provides previews in various formats for each.


<div> <div> <div><<</div> <div><</div> <div>/</div> <div>></div> <div>>></div> </div> <div>Results 1 to 1 (out of 1 items)</div> <div> <div>Search</div> </div> </div>				
Type	Name	Title	Common Formats	All Formats
	nurc:RunningExampleLayer	RunningExampleLayer	OpenLayers KML GML	<div>Select one ▼</div>
<div> <div> <div><<</div> <div><</div> <div>/</div> <div>></div> <div>>></div> </div> <div>Results 1 to 1 (out of 1 items)</div> </div>				

Figure 8: Previewing a Layer

The final GeoServer step is to preview the layer. The preview only shows the highest granularity of the aggregation index. Getting a preview of a layer is done as follows:

11. Click on the **Layer Preview** link in the **Data** section shown in Figure 1.
12. The **Layer Preview** page opens which displays all viewable layers like in Figure 8.
13. A preview format need to be selected from the drop-down menu highlighted in Figure 8.
14. Select a **WMS** preview format such as **PNG**.
15. A new web page will load (this might a few seconds depending on if the server side extension is enabled).
16. The final result will look like Figure 9.

The layer which shows the example dataset is now complete. The values of each square in the layer is calculated using the pre-aggregate index of the dataset. See 4.3 to learn how the layer can be used in combination with other tools such as OpenLayers¹⁰ to create a dynamic map which updates data on the fly.

0	0	0	0	0	1	7	1	0	1
0	0	0	0	140	1263	490	0	0	0
0	0	0	2	269	3119	1141	83	0	0
0	0	1	0	5	555	862	33	0	0
1	0	5	0	0	276	16	0	0	0
0	0	0	0	0	0	1	2	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	20	0	0	0
2	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figure 9: Preview of whole dataset

¹⁰<http://openlayers.org/>