Retail Project

Table Of Contents

1 Introduction

The Retail Black Friday project contains 5 major parts.

1. Cassandra CQL scripts to create the database schema, and CQL import scripts to populate it with seed data
2. An analytics Spark job with the source code available
3. A streaming data Spark job with the source code available
4. A sample Solr schema with a script to post it
5. A web project in the web-python directory - a simple Python / Flask project that runs against a local DSE node. This expects that you have a local DSE environment setup and running.
6. A web project in the web-java directory implemented with the same functionality as the web-python project.
7. The cash-register simulator implemented with JMeter

The Project is delivered in a fully configured VM, but can also be built manually from the GitHub repository. The instructions in Section 1 describe how to perform a manual installation. If you are working with the VM, proceed to Section 2.

2 Manual Installation – If you don't use the VM

If you would like to set up this project natively on a Mac or Linux computer, install all of the necessary components. If you are not installing on a Mac skip to the section “All Platforms – Pre-requisites”.

2.1 Mac Pre-requisites

1. Install brew - the default Mac python won't work
2. Install wget:

```
brew install wget
```

3. Install brew's version of python:

```
brew install python
```

4. Install pip:

```
brew install pip
```

Now continue with the instructions below to install the pre-requisites for the project.

2.2 All Platforms – Pre-requisites

1. Verify you can run python:

```
python
exit()
```

2. Verify you can run pip:

```
pip -V
```

3. Install Python dependencies:

```
pip install flask
pip install blist
pip install cassandra-driver
pip install requests
```

4. If you would like to use jupyter notebooks:

```
pip install jupyter
```

5. Install sbt

   For example, for Ubuntu:

```
echo "deb http://dl.bintray.com/sbt/debian /" | sudo tee -a
/etc/apt/sources.list.d/sbt.list
sudo apt-get update
sudo apt-get install sbt
```

6. Install JMeter

   Install our custom JMeter build:

```
wget https://s3.amazonaws.com/datastaxtraining/jmeter/apache-jmeter-
2.13.2.zip
```

   More information on the plugin can be found here: https://github.com/slowenthal/jmeter-cassandra

7. Download and install ActiveMQ

Apache ActiveMQ is an open source messaging server. JMeter will publish data to ActiveMQ queues. To do this, the `activeqm-all` jar file needs to be in the classpath of JMeter. Copy or link the jar to the jmeter/lib directory as shown below

a. Download the latest release from http://activemq.apache.org/activemq-5111-release.html
b. Create a directory in the location of your choice and untar the contents of the zip file.
c. Copy or link `activemq-all-5.11.1.jar` to `<path>/apache-jmeter/lib/activemq-all-5.11.1.jar`

NB Activemq functions as pipe between JMeter and Spark streaming and is a more reliable technology than using sockets. JMeter uses it's own JMS publisher sampler (which can both publish and receive) to publish a string to the HotProducts queue.

8. Install Java JDK 8

a. Java JDKs can be downloaded from

http://www.oracle.com/technetwork/java/javase/downloads/index.html

- Java JDK 8u45 can be downloaded directly from:

  http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

b. Run the JDK installer
c. Use the alternatives command to add a symbolic link to the Oracle JDK installation so that your system uses the Oracle JDK instead of the OpenJDK JRE.

For example, if your Oracle JDK has been installed in `/usr/java`:

```
# ls /usr/java
default  jdk1.7.0_67  jdk1.7.0_71  latest
```

The `alternatives` command will specify the default JDK to use:

```
 # alternatives --install /usr/bin/java java
/usr/java/jdk1.7.0_71/bin/java 200000
```

If more than one installation of java exists you can check that the correct version is in use:

```
# alternatives --config java
There are 6 programs which provide 'java'.
  Selection    Command
-----------------------------------------------
   1           /usr/lib/jvm/jre-1.7.0-openjdk.x86_64/bin/java
   2           /usr/lib/jvm/jre-1.6.0-openjdk.x86_64/bin/java
*  3           /usr/share/jdk_1.7.0.2
   4           /usr/share/jdk_1.7.0.2/bin/java
   5           /usr/share/jdk1.7
 + 6           /usr/share/jdk1.7/bin/java
```

```
Enter to keep the current selection[+], or type selection number: 6
```

Make sure your system is now using the correct JDK. For example:

```
# /usr/java/latest/bin/java -version
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b14)
Java HotSpot(TM) 64-Bit Server VM (build 24.71-b04, mixed mode)
```

9. Install & configure DSE

   a. Download the latest DSE release from www.datastax.com/downloads
   b. Create a single node cluster on the machine where the repo has been downloaded.

   As a minimum set the following parameters in Cassandra.yaml:

   - Listener address: 127.0.0.1
   - RPC address: 127.0.0.1
   - Set the data file locations with correct write permissions:
     - Cassandra data files `/var/lib/cassandra`
     - Spark data file location to `/var/lib/spark`
   - Set the log file locations with correct write permissions:
     - Cassandra log files `/var/log/cassandra`
     - Spark log files `/var/log/spark`

   NB We will need to run DSE with Spark and Cassandra simultaneously. In a packaged install you can configure DSE to start with Spark and Solr using `/etc/default/dse`

   For tarball installs you use both the -s and -k flags when starting Cassandra to enable both Solr and Spark on the same node.

10. If you would like to run jupyter notebooks to help with spark development

```
pip install ipython
pip install notebook
```

Download and install the latest release from here according to the instructions:

```
https://github.com/slowenthal/spark-kernel/releases
```

2.3 Manual Installation – Configure The Project

The Retail project is designed to run on a single node configured for Cassandra, Spark, Solr and JMeter. This is the default configuration. If you will be using this configuration you can proceed to the section "How To Use The Project".

Alternatively if you intend to run JMeter and the web components against a remote node running Cassandra, Spark and Solr you will need to adjust the configuration to set the correct destination IP address in the following files:

1. jmeter/scans.jmx
2. web-python/application.cfg
3. web-java/application.cfg

3 VM Installation

The pre-built and configured VM can be downloaded from here:

https://s3.amazonaws.com/datastaxtraining/VM/Capstone-VM-1.0.zip

a. When the download has completed, unzip the contents into a location of your choice. This will create a directory called “Retail Demo” containing a file called “Retail Demo.vbox”
b. Open the vbox file using Virtual Box to create the VM
   - Under the Machine menu select the add option and choose the vbox file provided
c. Ensure that you have at least 8GB and 3 CPU allocated to the VM
d. Start the VM – no login credentials are required
e. Ignore any warning messages relating to missing shared folders

4 How To Use The Project

The instructions below assume that you have either:

1. Created an environment containing all the pre-requisites and can successfully start Cassandra, Spark, Solr and JMeter, or
2. Downloaded and started the VM

Regardless of the machine type you will be using you must ensure that you have sufficient resources available to Spark Streaming, particularly CPU, but also memory. In oparticular, Spark Streaming requires 2 Spark Worker cores.

1. CPU – you will need to allocate at least three CPUs to your server environment by uncommenting and set SPARK_WORKER_CORES in `spark-env.sh` (in `<DSE_HOME>/resources/spark/conf`):

```
# Set the number of cores used by Spark Worker - if uncommented,
it overrides the setting initial_spark_worker_resources in
dse.yaml.
export SPARK_WORKER_CORES=3
```

2. Memory – we recommend that you allocate 8GB to your server environment. If you have less RAM available you may need to tweak SPARK_WORKER_MEMORY in `spark-env.sh` (in `<DSE_HOME>/resources/spark/conf`):

```
# Set the amount of memory used by Spark Worker - if uncommented,
it overrides the setting initial_spark_worker_resources in
dse.yaml.
# export SPARK_WORKER_MEMORY=2048m
```

4.1 Start the DSE services – Cassandra, Spark and Solr

a. Tarball install (manual install):

```
dse cassandra -k -s
```

The services can be stopped using:

```
dse cassandra-stop
```

b. Package install (VM)

```
service dse start  * Starting DSE daemon dse
DSE daemon starting with Solr enabled (edit /etc/default/dse to disable)
DSE daemon starting with Spark enabled (edit /etc/default/dse to disable)[ OK ]
```

NB the services will be autostarted when the VM is started. Use the following command to restart at any point:

```
service dse restart  *
```

4.2 Import the seed data into Cassandra

a. Navigate to the `<retail project>/cql` directory:

```
cd cql
```

b. Create the retail keyspace and tables:

```
cqlsh -f retail.cql
```

c. Import the seed data:

```
cqlsh -f import.cql
```

Note: in DSE 4.7.0, there is a bug which prevents the above command from running correctly. In that release, run the import as follows:

```
cat import.cql | cqlsh
```

This script will populate the following tables:

- `suppliers`
- `products_by_id`
- `products_by_supplier`
- `products_by_category_name`
- `stores`

It will also define the following hot products:

```
UPDATE retail.products_by_category_name SET is_hot = true WHERE
category_name = 'notebooks';
UPDATE retail.products_by_category_name SET is_hot = true WHERE
category_name = 'servers';
```

4.3 Import the sample Solr index

a. Navigate to the `<retail project>/solr` directory:

```
cd solr
```

b. Run the script to import the solr index:

```
./add-schema.sh products_by_id
```

Ensure that you see these three messages:

```
SUCCESS
SUCCESS
Created index.
```

4.4 Start the Python web project

The Python-based web services run as a Flask lightweight web framework.

a. Navigate to the `<retail project>/web-python` directory:

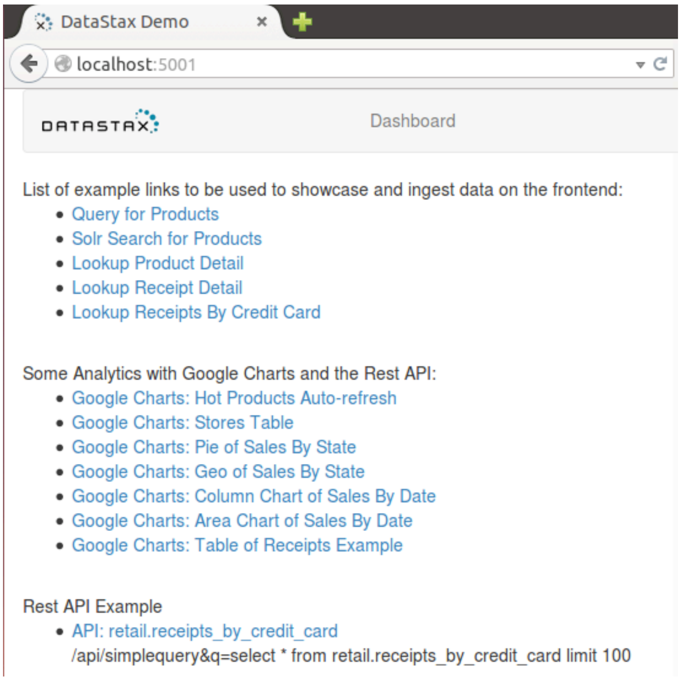
```
cd web-python
```

b. Start Flask web services in the background:

```
./run &
```

4.5 Verify that the Python web project is running

In the your browser navigate to the URL http://localhost:5001/ - you should see the following displayed:



List of example links to be used to showcase and ingest data on the frontend:

- Query for Products
- Solr Search for Products
- Lookup Product Detail
- Lookup Receipt Detail
- Lookup Receipts By Credit Card

Some Analytics with Google Charts and the Rest API:

- Google Charts: Hot Products Auto-refresh
- Google Charts: Stores Table
- Google Charts: Pie of Sales By State
- Google Charts: Geo of Sales By State
- Google Charts: Column Chart of Sales By Date
- Google Charts: Area Chart of Sales By Date
- Google Charts: Table of Receipts Example

Rest API Example

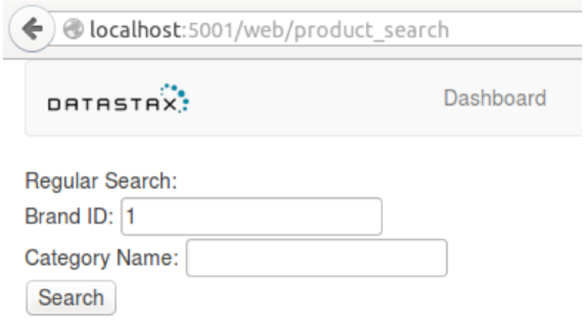
- API: retail.receipts_by_credit_card
  /api/simplequery&q=select * from retail.receipts_by_credit_card limit 100

Try out the pages that return the available seed data.

NB the “Lookup Receipt Detail”, “Lookup Receipts By Credit Card”, “Hot Product”, “Sales by State” and “Sales by Date” options will not return any data at this point as the underlying tables are not yet populated - this happens in the following steps.

a. Query For Products

Select “Query for Products” – enter (e.g. “1”) in the Brand ID field and click Search:



localhost:5001/web/product_search

DATASTAX Dashboard

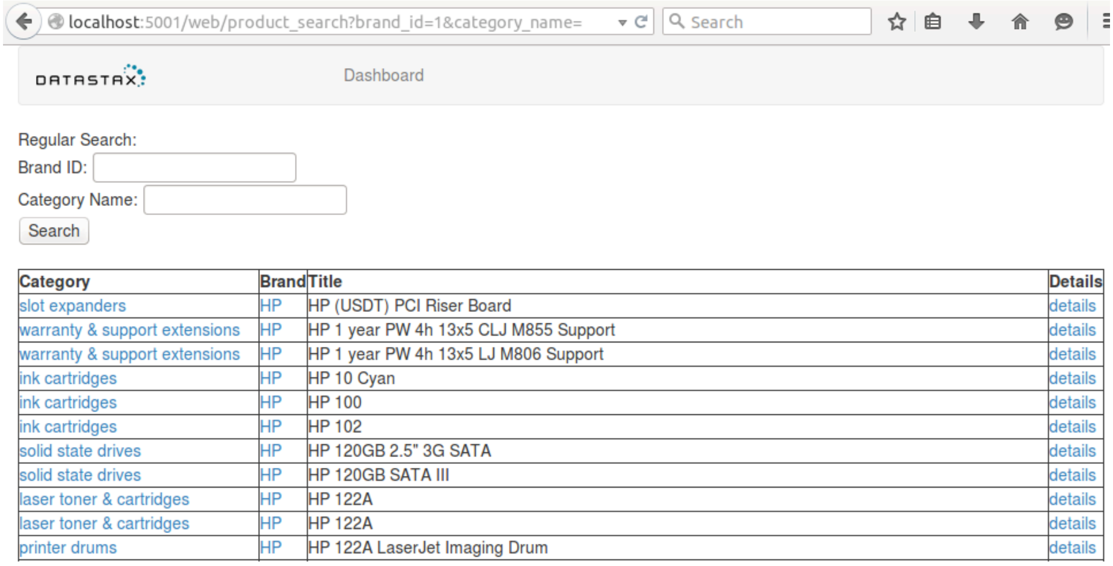
Regular Search:

Brand ID: 1

Category Name:

Search

You will see a selection of products displayed for the product code entered:



localhost:5001/web/product_search?brand_id=1&category_name=

DATASTAX Dashboard

Regular Search:

Brand ID:

Category Name:

Search

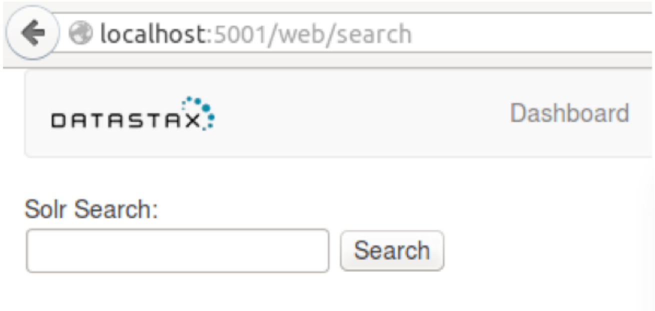
Category	Brand	Title	Details
slot expanders	HP	HP (USDT) PCI Riser Board	details
warranty & support extensions	HP	HP 1 year PW 4h 13x5 CLJ M855 Support	details
warranty & support extensions	HP	HP 1 year PW 4h 13x5 LJ M806 Support	details
ink cartridges	HP	HP 10 Cyan	details
ink cartridges	HP	HP 100	details
ink cartridges	HP	HP 102	details
solid state drives	HP	HP 120GB 2.5" 3G SATA	details
solid state drives	HP	HP 120GB SATA III	details
laser toner & cartridges	HP	HP 122A	details
laser toner & cartridges	HP	HP 122A	details
printer drums	HP	HP 122A LaserJet Imaging Drum	details

Try the Category Name search by entering one of the values returned in the search above (e.g. “slot expanders”).

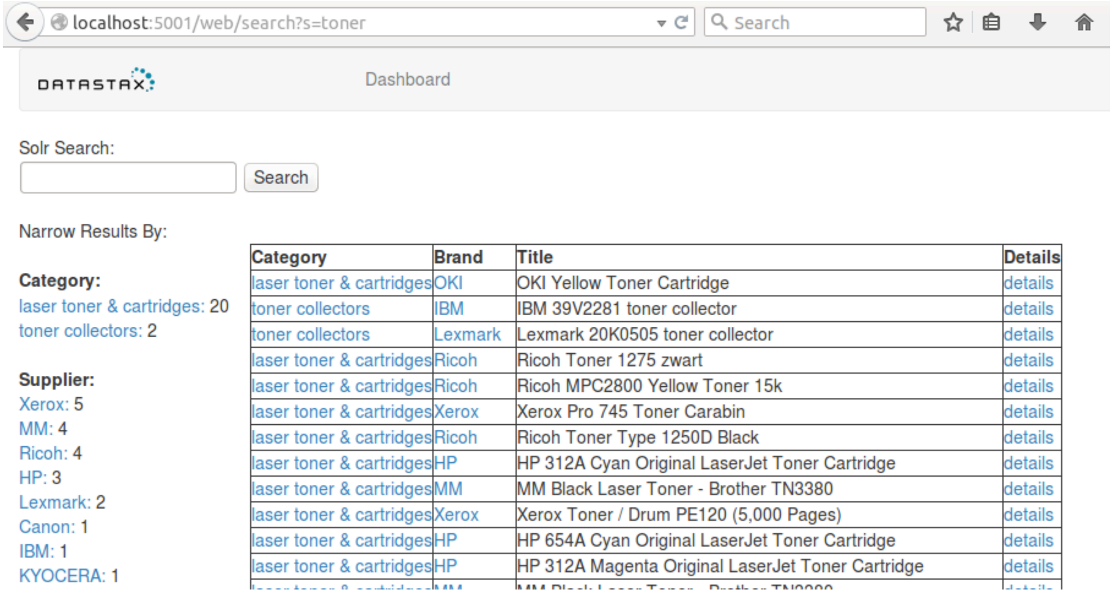
Note that you can also follow the hyperlinks of the displayed products (e.g. click “IBM” to see all the IBM products).

b. Solr Search for Products

Select “Solr search for Products” – enter (e.g. “toner”) in the Solr Search field and click Search:



You will see a selection of products displayed for the product type entered:



Narrow Results By:

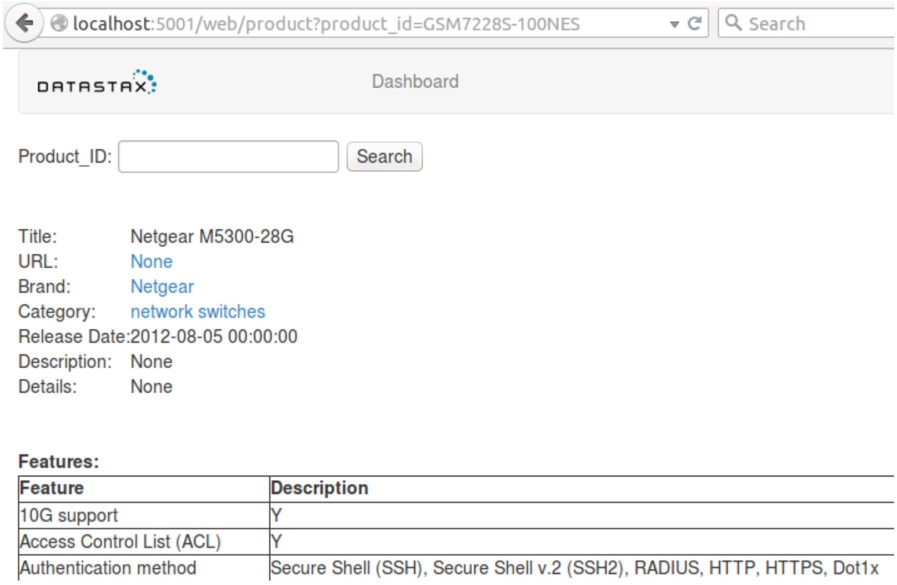
Category:
laser toner & cartridges: 20
toner collectors: 2

Supplier:
Xerox: 5
MM: 4
Ricoh: 4
HP: 3
Lexmark: 2
Canon: 1
IBM: 1
KYOCERA: 1

Category	Brand	Title	Details
laser toner & cartridges	OKI	OKI Yellow Toner Cartridge	details
toner collectors	IBM	IBM 39V2281 toner collector	details
toner collectors	Lexmark	Lexmark 20K0505 toner collector	details
laser toner & cartridges	Ricoh	Ricoh Toner 1275 zwart	details
laser toner & cartridges	Ricoh	Ricoh MPC2800 Yellow Toner 15k	details
laser toner & cartridges	Xerox	Xerox Pro 745 Toner Carabin	details
laser toner & cartridges	Ricoh	Ricoh Toner Type 1250D Black	details
laser toner & cartridges	HP	HP 312A Cyan Original LaserJet Toner Cartridge	details
laser toner & cartridges	MM	MM Black Laser Toner - Brother TN3380	details
laser toner & cartridges	Xerox	Xerox Toner / Drum PE120 (5,000 Pages)	details
laser toner & cartridges	HP	HP 654A Cyan Original LaserJet Toner Cartridge	details
laser toner & cartridges	HP	HP 312A Magenta Original LaserJet Toner Cartridge	details
laser toner & cartridges	MM	MM Black Laser Toner - Brother TN[illegible]	details

c. Lookup Product Detail

You can find product ID’s by querying the product_id table in cqlsh. For example, search for “GSM7228S-100NES”:



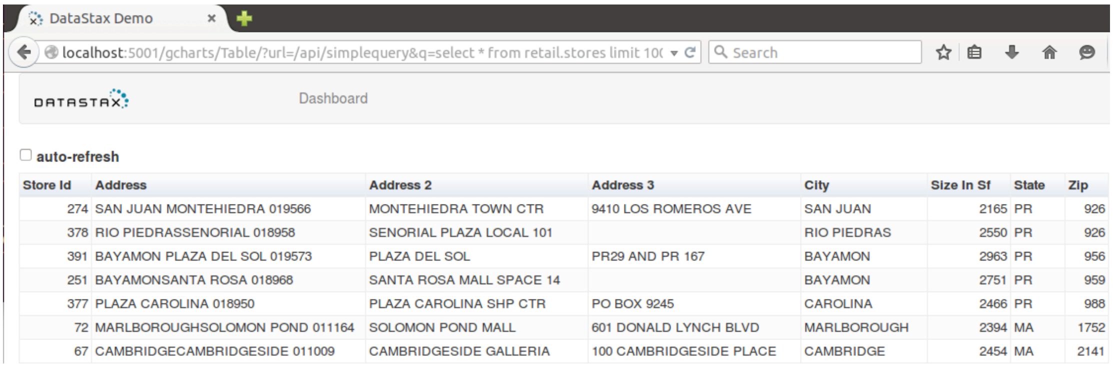
Title: Netgear M5300-28G
URL: None
Brand: Netgear
Category: network switches
Release Date:2012-08-05 00:00:00
Description: None
Details: None

Features:

Feature	Description
10G support	Y
Access Control List (ACL)	Y
Authentication method	Secure Shell (SSH), Secure Shell v.2 (SSH2), RADIUS, HTTP, HTTPS, Dot1x

d. Google Charts: Stores Table

This is a query of the Stores table with the results presented in a table by Google Charts:



DataStax Demo
localhost:5001/gcharts/Table/?url=/api/simplequery&q=select * from retail.stores limit 10(
Search
DATASTAX
Dashboard

□ auto-refresh

Store Id	Address	Address 2	Address 3	City	Size In Sf	State	Zip
274	SAN JUAN MONTEHIEDRA 019566	MONTEHIEDRA TOWN CTR	9410 LOS ROMEROS AVE	SAN JUAN	2165	PR	926
378	RIO PIEDRASSENORIAL 018958	SENORIAL PLAZA LOCAL 101		RIO PIEDRAS	2550	PR	926
391	BAYAMON PLAZA DEL SOL 019573	PLAZA DEL SOL	PR29 AND PR 167	BAYAMON	2963	PR	956
251	BAYAMONSANTA ROSA 018968	SANTA ROSA MALL SPACE 14		BAYAMON	2751	PR	959
377	PLAZA CAROLINA 018950	PLAZA CAROLINA SHP CTR	PO BOX 9245	CAROLINA	2466	PR	988
72	MARLBOROUGHSOLOMON POND 011164	SOLOMON POND MALL	601 DONALD LYNCH BLVD	MARLBOROUGH	2394	MA	1752
67	CAMBRIDGECAMBRIDGESIDE 011009	CAMBRIDGESIDE GALLERIA	100 CAMBRIDGESIDE PLACE	CAMBRIDGE	2454	MA	2141

4.6 Simulate the Cash Registers Using JMeter

Transactions and receipts are generated using JMeter.

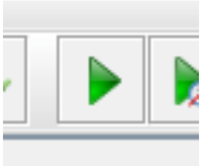
a. Navigate to the `<retail project>/jmeter` directory:

```
cd jmeter
```

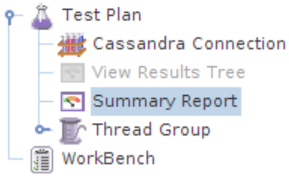
b. The JMeter executable is already in your path and can be started in the foreground using the GUI interface using the following command:

```
Jmeter -t scans.jmx
```

In the GUI you must start the JMeter data injection by pressing the start button:



You can track the activity during injection by clicking on Summary Report:



NB A small percentage of JMeter errors is acceptable (this is due to variations in the source data format) but high error levels indicate that there is a problem. However the JMS Publisher will always error when activemq is not running (you will start this in a later step for the Hot Products queue).

NB You can run JMeter in text mode by starting it with the –n option e.g.:

```
jmeter –n -t scans.jmx
```

You can also log output to a log file from the command line using the –l option e.g.:

```
jmeter -t scans.jmx -l scans.log
```

NB JMeter will insert a significant amount of data, so run it sparingly. A 1-2 hour run should be sufficient.

JMeter will insert records into the following tables;

- `receipts`
- `inventory_per_store`
- `receipts_by_credit_card`
- `receipts_by_store_date`

4.7 Build the Spark Analytics job

The Spark analytics job will roll up transactions into the analytics tables used by the charts:

- `sales_by_date`
- `sales_by_state`

a. Navigate to the `<retail project>/spark` directory:
b. Run sbt assembly from the prompt – it may take a few seconds to download required files and compile - this will compile any source files found in src/main/scala (RollupRetail.scala):

```
$ sbt assembly
…
[info] Done updating.
```

```
[info] Set current project to spark-retail (in build
file:/home/dse/retail/spark/)
[info] Packaging /root/retail/spark/target/scala-2.10/spark-retail-
assembly-1.0.jar ...
[info] Done packaging.
[success] Total time: 164 s, completed May 1, 2015 2:42:24 PM
```

c. You have now created the Spark jar file - exit sbt:

```
> exit
```

4.8 Submit the Spark Analytics job

This job will extract data from Cassandra, rollup the data and populate the analytics tables.

a. Navigate to the `<retail project>/spark` directory:
b. Submit the analytics job:

```
dse spark-submit --class RollupRetail target/scala-2.10/spark-retail-
assembly-1.0.jar
```

You could put this in a shell script for convenience (eg. run_spark.sh).

NB The following error message suggests that the spark workers have died (this can happen for example when you close the laptop) or that there is insufficient memory.

```
15/05/06 14:43:35 WARN TaskSchedulerImpl: Initial job has not accepted
any resources; check your cluster UI to ensure that workers are
registered and have sufficient memory
```

a. Check that you have 8GB allocated to the machine
b. Try restarting the spark workers using the command:

```
dsetool sparkworker restart
```

c. If that fails to resolve the problem, stop and restart DSE products:

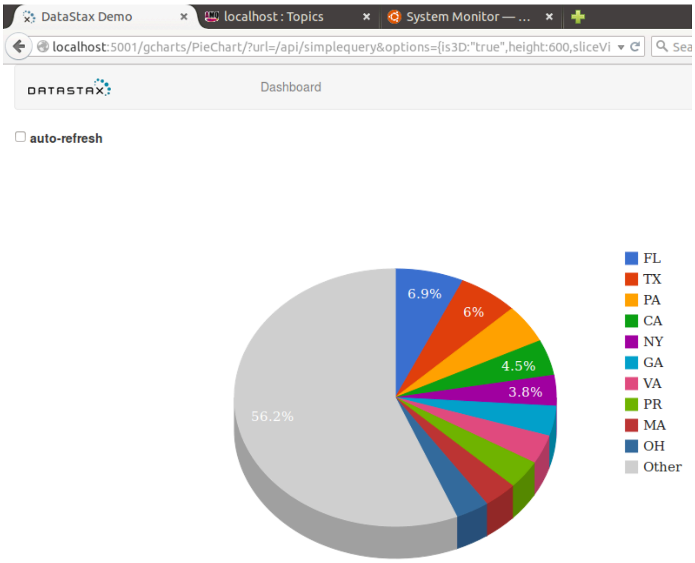
```
dse cassandra-stop
dse cassandra –k -s
```

d. If you are unable to resolve the problem using the methods above, restart the virtual machine (there is a lot of contention for resources when running DSE, Spark and Solr on the same machine).

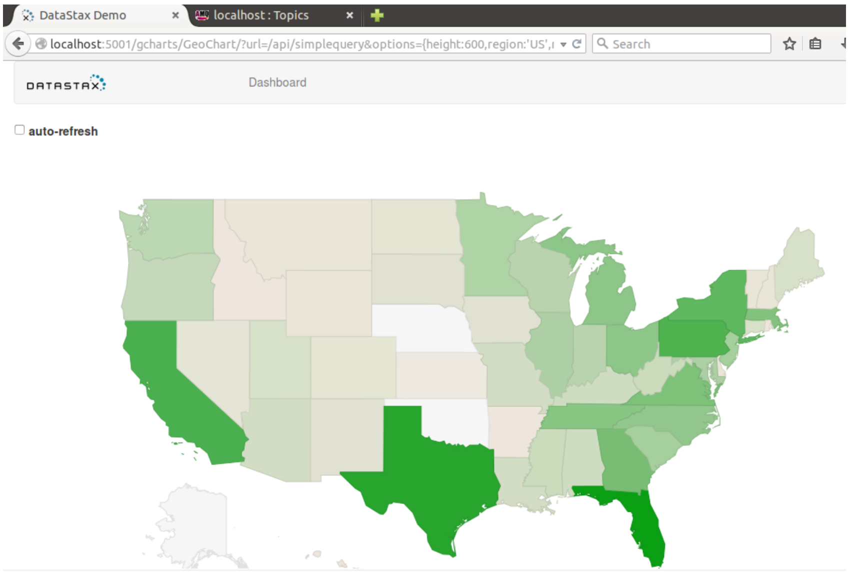
4.9 View the Charts

As data is progressively injected into Cassandra by JMeter you will be able to see more data reflected in the charts. Examples are shown below.

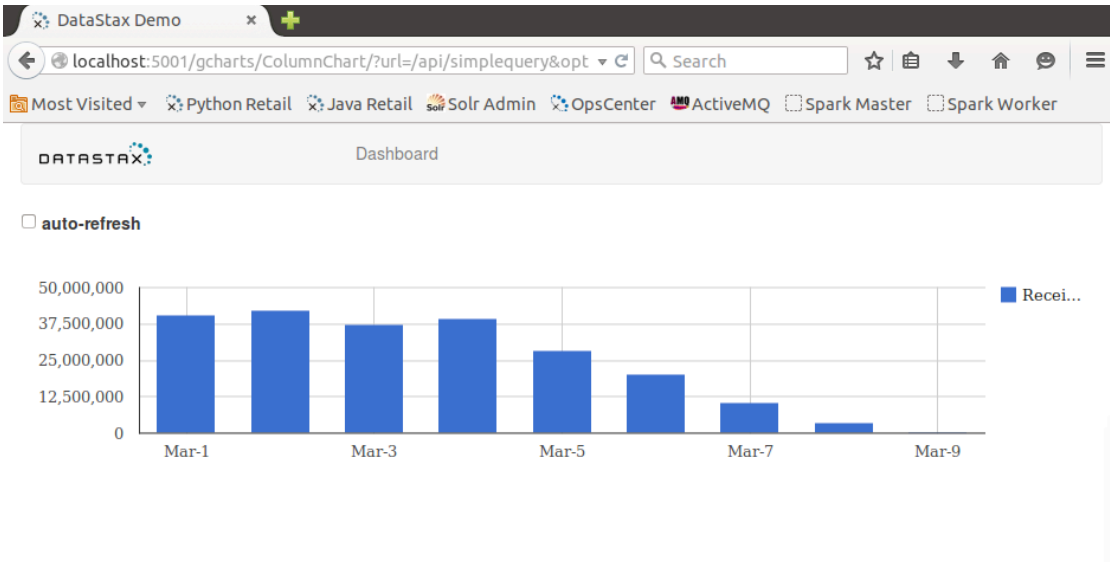
a. Pie of Sales By State (JMeter injection data)



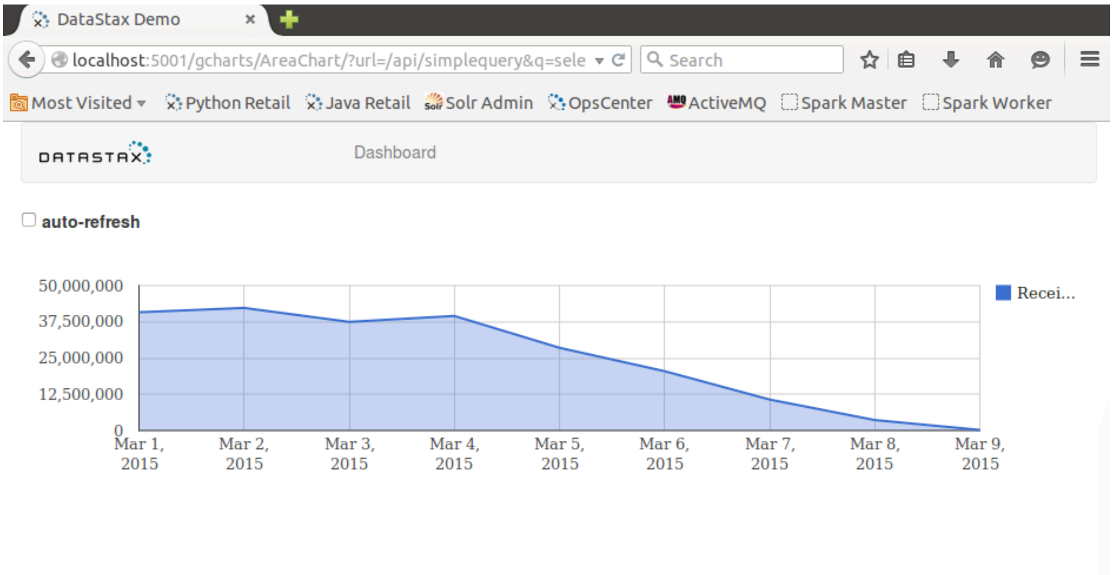
b. Geo of Sales By State (JMeter injection data)


DataStax Demo
localhost : Topics
localhost:5001/gcharts/GeoChart/?url=/api/simplequery&options={height:600,region:'US',
Search
DATASTAX
Dashboard
□ auto-refresh

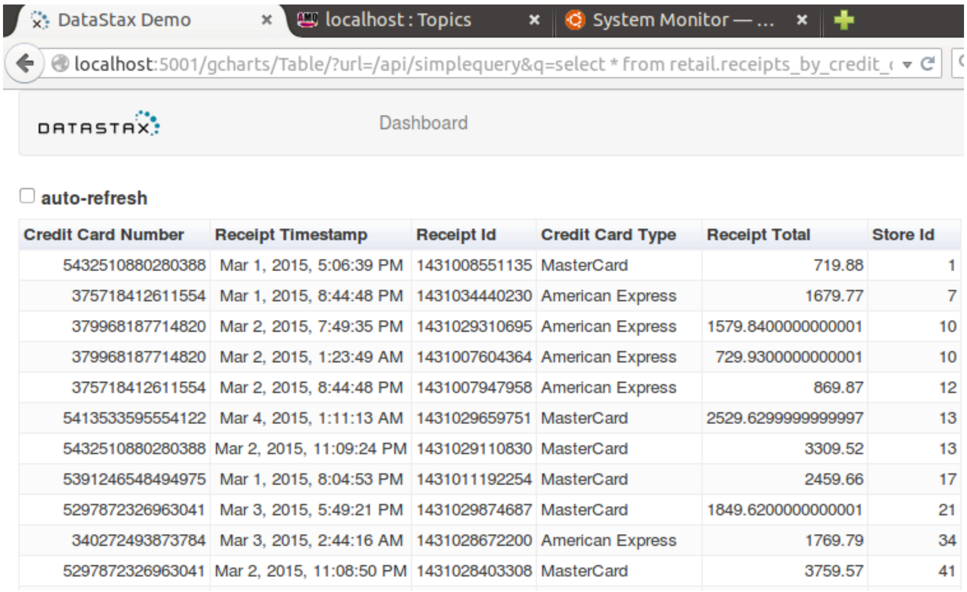
c. Column Chart of Sales By State (Spark Analytics rollup data)



d. Area Chart of Sales By State (Spark Analytics rollup data)



e. Table of Receipts Example (JMeter injection data)



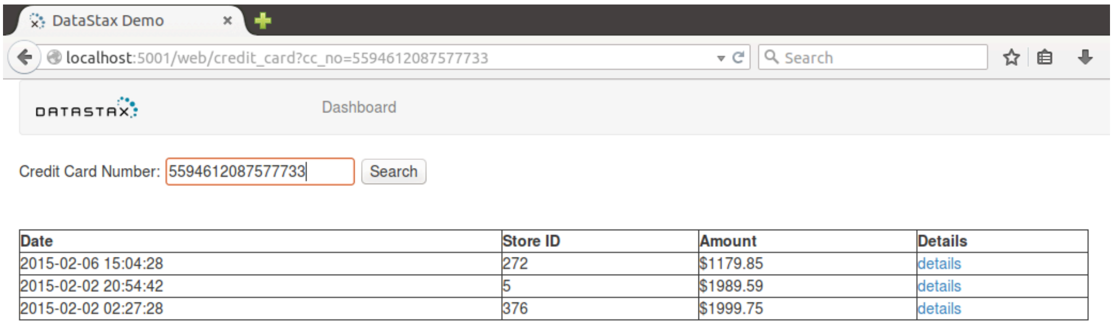
DataStax Demo
localhost : Topics
System Monitor — ...
localhost:5001/gcharts/Table/?url=/api/simplequery&q=select * from retail.receipts_by_credit_
DATASTAX
Dashboard

□ **auto-refresh**

Credit Card Number	Receipt Timestamp	Receipt Id	Credit Card Type	Receipt Total	Store Id
5432510880280388	Mar 1, 2015, 5:06:39 PM	1431008551135	MasterCard	719.88	1
375718412611554	Mar 1, 2015, 8:44:48 PM	1431034440230	American Express	1679.77	7
379968187714820	Mar 2, 2015, 7:49:35 PM	1431029310695	American Express	1579.8400000000001	10
379968187714820	Mar 2, 2015, 1:23:49 AM	1431007604364	American Express	729.9300000000001	10
375718412611554	Mar 2, 2015, 8:44:48 PM	1431007947958	American Express	869.87	12
5413533595554122	Mar 4, 2015, 1:11:13 AM	1431029659751	MasterCard	2529.6299999999997	13
5432510880280388	Mar 2, 2015, 11:09:24 PM	1431029110830	MasterCard	3309.52	13
5391246548494975	Mar 1, 2015, 8:04:53 PM	1431011192254	MasterCard	2459.66	17
5297872326963041	Mar 3, 2015, 5:49:21 PM	1431029874687	MasterCard	1849.6200000000001	21
340272493873784	Mar 3, 2015, 2:44:16 AM	1431028672200	American Express	1769.79	34
5297872326963041	Mar 2, 2015, 11:08:50 PM	1431028403308	MasterCard	3759.57	41

f. Receipts by Credit Card

The credit card transaction data can now be used in the Receipts by Credit Card page.



Credit Card Number: 5594612087577733

Date	Store ID	Amount	Details
2015-02-06 15:04:28	272	$1179.85	details
2015-02-02 20:54:42	5	$1989.59	details
2015-02-02 02:27:28	376	$1999.75	details

4.10 Build the Spark Streaming job

a. Navigate to the `<retail project>/sparkstreaming` directory:

b. Run sbt from the prompt – it may take a few seconds to download required files:

```
$ sbt
```

c. When it finishes updating you will be returned to the prompt:

```
[info] Done updating.
[info] Set current project to spark-retail (in build
file:/home/dse/retail/spark/)
>
```

d. Run the assembly command – when it finishes exit from sbt:

```
[info] Packaging /root/retail/sparkstreaming/target/scala-2.10/spark-streaming-retail-assembly-1.0.jar ...
[info] Done packaging.
[success] Total time: 20 s, completed May 8, 2015 2:56:19 PM
> exit
```

4.11 Start activemq

The Spark Streaming job receives data that JMeter publishes to the Hot Products queue managed by activemq. The activemq process must be running to receive the hot products from JMeter:

```
cd ~/activemq/bin
./activemq start
```

4.12 Submit the Spark Streaming job

a. Navigate to the `<retail project>/sparkstreaming` directory:

b. Submit the streaming job:

The Spark Streaming job will run continually, reading hot product events placed on the activemq message queue, and populate the table:

- `real_time_analytics`

```
dse spark-submit --class HotProductsStream target/scala-2.10/spark-streaming-retail-assembly-1.0.jar
```

NB Ensure that JMeter is running the data injection when the streaming job is sumitted.

c. View the Hot Products queue

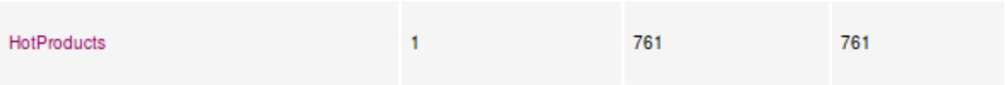
Navigate to the Hot Products web page. Click on “auto-refresh” to see the products updated in real-time.

Updates to the hot products page will occur in real-time e.g. if JMeter stops publishing hot products to the message queue, then so does the hot products display

Login in the activemq console as admin with the password admin to view the queue status:

http://localhost:8161/admin/topics.jsp

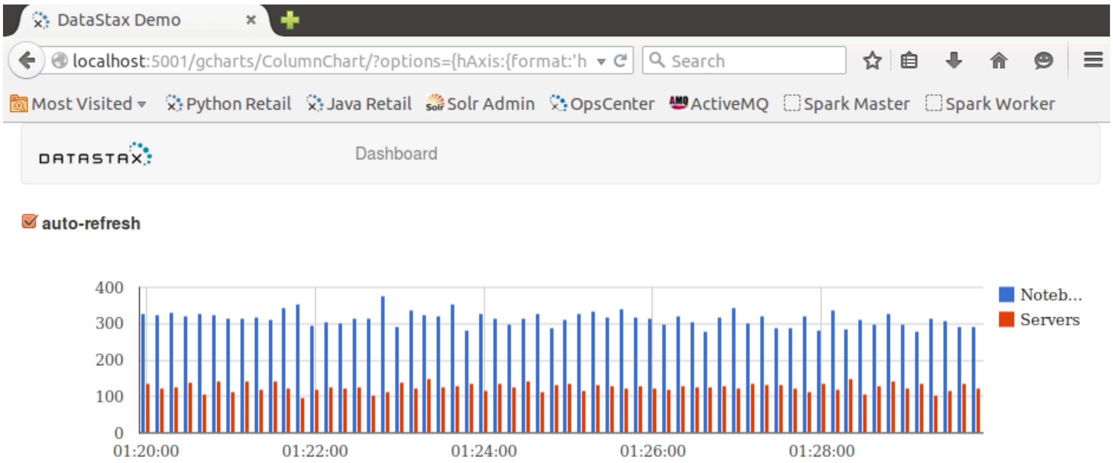
The topics tab will show a count of items published to the HotProducts queue for example



HotProducts	1	761	761

d. View the Hot Products page

Visit the Hot Products chart, and watch the sales of notebooks and servers.



NB you can change the products classed as "hot products" by changing the is_hot boolean on the products_by_category_name table. The JMeter script will also randomly update product "Hot Product" status.

4.13 Starting Jupyter Notebooks for Scala (Spark-Cassandra)

a. Navigate to the jupyter directory
b. Run

```
ipython notebook --port 7001
```

4.14 Starting Jupyter Notebooks for Pyspark

Note: If you start Jupyter using these instructions, the Spark Kernel will run not run correctly even though it appears to be available .

c. Navigate to the jupyter directory
d. Set environment variables as follows:

```
export PYSPARK_DRIVER_PYTHON=ipython
export PYSPARK_DRIVER_PYTHON_OPTS="notebook --port 7001"
```

e. Start pyspark under dse. The notebook should pop up

```
dse pyspark
```

5 Using the Java Web Framework

The Java web framework is provided as an alternative to Python to showcase development against Cassandra in Java. Version 8 of Java is required.

a. Navigate to the `<retail project>/web-java` directory:

b. If you are installing from scratch, or need to rebuild the project, run maven from the prompt – it may take a few seconds to download required files before successfully completing the build:

```
mvn install
```

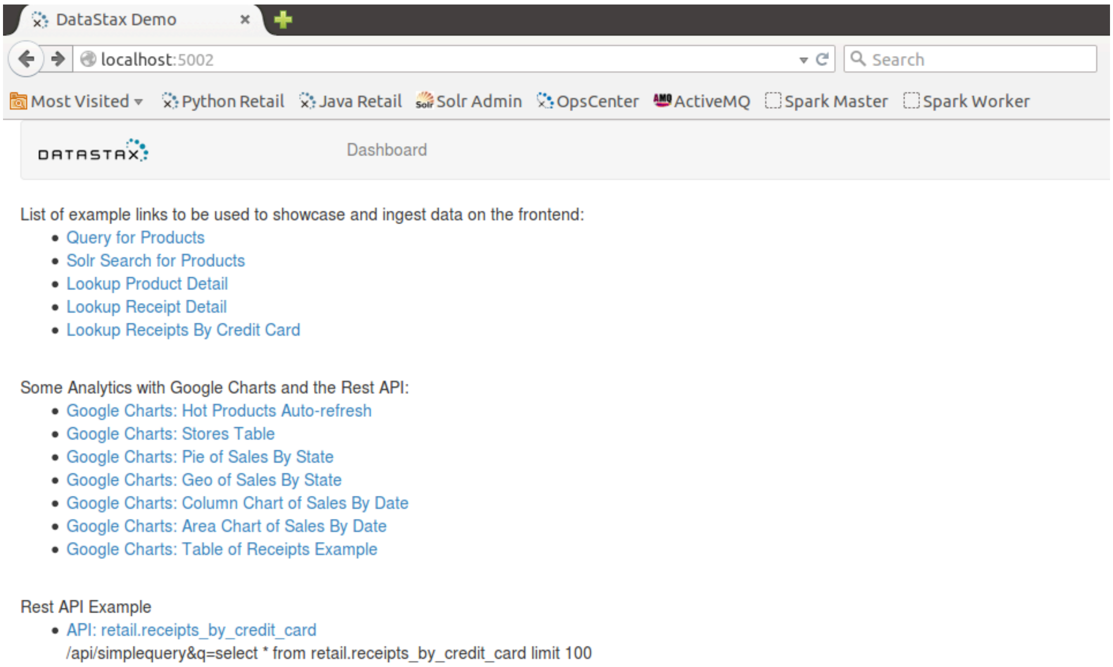
```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS [INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 13.207s [INFO] Finished at: Mon May 11 14:10:10 UTC 2015
[INFO] Final Memory: 15M/217M
[INFO] ------------------------------------------------------------------------
```

c. Run the compiled code:

```
java -cp "target/retail-1.0.jar:target/lib/*" StartJetty
```

5.1 Verify that the Java web project is running

In the your browser navigate to the URL http://localhost:5002/ - you should see the following displayed:



List of example links to be used to showcase and ingest data on the frontend:

- Query for Products
- Solr Search for Products
- Lookup Product Detail
- Lookup Receipt Detail
- Lookup Receipts By Credit Card

Some Analytics with Google Charts and the Rest API:

- Google Charts: Hot Products Auto-refresh
- Google Charts: Stores Table
- Google Charts: Pie of Sales By State
- Google Charts: Geo of Sales By State
- Google Charts: Column Chart of Sales By Date
- Google Charts: Area Chart of Sales By Date
- Google Charts: Table of Receipts Example

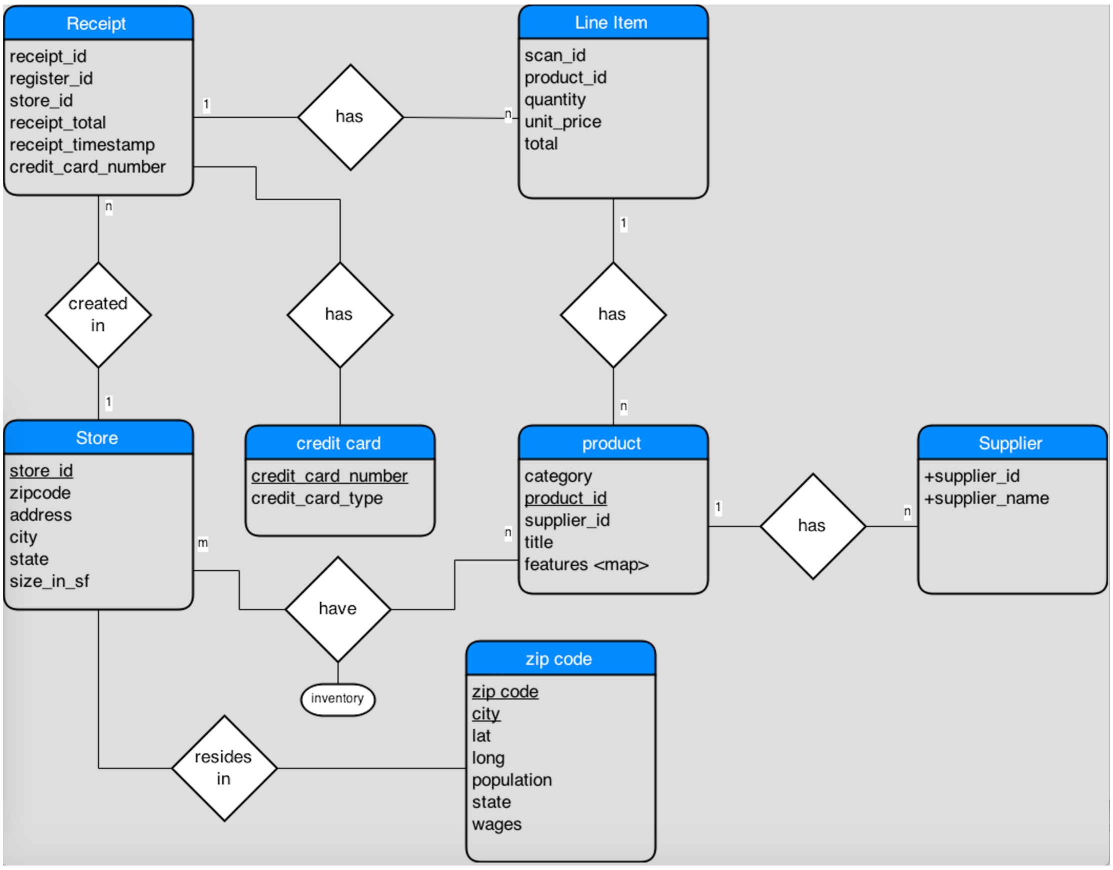
Rest API Example

- API: retail.receipts_by_credit_card
  /api/simplequery&q=select * from retail.receipts_by_credit_card limit 100

5.2 Test the web pages

Follow the instructions for the Python web project to check the functionality of the Java web project pages.

6 Appendix 1. – Conceptual Schema Reference



7 Appendix 2. – Schema Table Reference

7.1 Seeded Tables

The following tables are populated when the CQL import script is run:

- suppliers
- products_by_id
- products_by_supplier
- products_by_category_name
- stores

7.2 Jmeter Tables

The following tables are populated when the JMeter simulation is run:

- receipts
- inventory_per_store
- receipts_by_credit_card
- receipts_by_store_date

7.3 Spark Rollup Tables

The following tables are populated when the Spark Analytics job is run:

- sales_by_date
- sales_by_state

7.4 Spark Streaming Tables

The following table is populated when the Spark Streaming job is run:

- real_time_analytics

NB This is an unusual table as it contains timestamp as a clustering column, and quantity is a map of products and the quantities sold. The map is dynamic as in any time period the set of hot products can change.

7.5 Unused Tables

The following table is not currently used:

- Zipcodes

8 Appendix 3. Capstone Web Application

The Capstone base contains the same sample web application in both python and Java.

- Some sample web pages using the Jinja web templates
- A sample web template that contains a google chart
- A rest API with 2 apis that can retrieve Cassandra data and return it in a format which works for google chart.
- A very light MVC Web framework

Internals of each one:

The applications are divided into the following components as diagrammed below. In addition, there is a description of how these components are implemented in Java and Python.

8.1 View - Jinja Web Templates

Both the Java and Python applications use jinja template. On Python, we are using Flask as our web framework, and Jinja2 is an integral part of it. Not wanting to rewrite the templates in a different language, the Java app includes jinjava to process Jinja2 templates. The templates have been carefully crafted as to keep them common between the 2 platforms. To do this we:

- Did not use any language-specific syntax
- Stuck to standard relative paths for included templates, and URLs eg. /base.jinja2
- Created and registered a single helper function in both languages called makeURL as opposed to adding any fancy language code.

For Java, we also treat the JSON output like a view, and have a method to convert a Cassandra ResultSet to json in the correct format.

Model - We skipped these in python, but implemented it in Java. The model objects represent entities in the data model, and the operations on it. An entity may involve one or more Cassandra tables.

Controllers - controllers handle the flow of the application. They receive a web request, call the model to retrieve or store data, and then return the results or the next web page. In the case of the rest api, they simply return the JSON result. For the regular web pages, they render the appropriate Jinja2 templates. The frameworks each have their own way of mapping URLs to the appropriate method or function to server that URL.

8.2 Some Helper Classes

Cassandra Object - This manages the connections and sessions to Cassandra. It implements a single connection, and it is assumed the nodes in that data center are adequate for both the Cassandra workload and Solr workload.

JinjaHelper - Functions that are used in the jinja templates.

jsonoutput - Functions to assist with building json for google charts. Not used in Python.

The Mainline - This configures the web application framework and starts it. It reads the application.cfg file and configures the port, the Cassandra connection, and Jinja helpers, etc.

9 How the charts work

The google charts are generated by a single page (/gcharts) which takes a few parameters

- Chart Type
- A REST API call which returns a data table in JSON in a format friendly to Google Charts
- Parameters to REST API
- Google Chart options

```
http://localhost:5001/gcharts/PieChart/?url=/api/simplequery&optio
ns={height:600,sliceVisibilityThreshold:.03}&q=select%20state,%20r
eceipts_total%20from%20sales_by_state
```

Many charts may be created purely by manipulating the URL. The parameters options, url, and sort order are recognized by the chart API. All others are simply tagged on to the url. The tagged url parameter and options are passed to the template. The resultant web page makes an ajax call to the URL to retrieve the data, and then draws the chart. The options are passed to the Google chart API verbatim.

10 Jinja Template Details (Java & Python)

Jinja2 is a simple templating language used in this this project for rending dynamic web content. To render a web page, simply call the rendering function, and pass in all of the parameters you reference in the template. There are examples in each platform of rendering the templates.

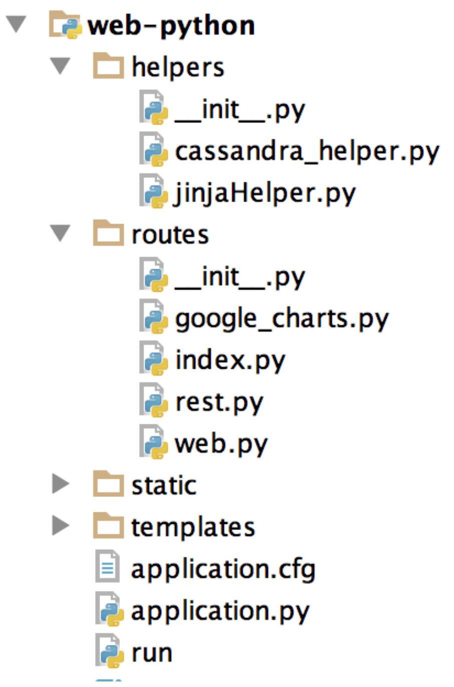
The project just uses a couple of simple features of the jinja templates.

- {% <some code> %} - we use some simple code that is supported in jinjava. For java, it's a bit restrictive, whereas for python the block may contain any python expressions. Some simple statements are for, set, if, and a method call

- {{ <variable expression }} - This simply outputs the given expression. In Java, you can dereference fields in beans so long as the getter is in CamelCase. If the bean as getProductName(), we can have expression in the template like {{ product.product_name }}

11 Python (Flask) Implementation

11.1.1 Code Tree



```
web-python
    helpers
        __init__.py
        cassandra_helper.py
        jinjaHelper.py
    routes
        __init__.py
        google_charts.py
        index.py
        rest.py
        web.py
    static
    templates
    application.cfg
    application.py
    run
```

Mainline

Application.py - This registers python objects to serve URL patterns, and registers the makeURL function to Jinja2.

```
from utils.JinjaHelper import makeURL
from routes import rest
from routes import web
from routes.index import index_api
from routes.rest import rest_api
from routes.google_charts import gcharts_api
from routes.web import web_api
```

```
app.register_blueprint(index_api)
app.register_blueprint(rest_api, url_prefix='/api')
app.register_blueprint(gcharts_api, url_prefix='/gcharts')
app.register_blueprint(web_api, url_prefix='/web')
```

Registering new routes (URL to python function):

Routes are registered in 2 phases:

If you are creating a new python file, register it in the file as follows: Creating a new Blueprint for the file:

```
web_api = Blueprint('web_api', __name__)
```

Then Give the file a URL pattern and register it in the mainline (Application.py)

```
app.register_blueprint(web_api, url_prefix='/web')
```

1) In the file you have created or are extending, create functions for next part of the URL. You may have parameterized parts of the URL pattern in the case you are implementing REST-style URLs. In the example below, <series> is a parmeter.

```
@rest_api.route('/realtime/<series>')
def timeslice(series=None):
```

Implementing the function:

If you have any query parameters on the URL, retrieve them with the request.args.get method. This example gets the minutes query. If the URL is <url>?minutes=....

```
request.args.get('minutes', 5)
```

If the query string is not present, it will default to 5. Then implement your logic to store and retrieve information from Cassandra, and then render and return the result. This example renders a template passing product as a parameter:

```
return render_template('product_list.jinja2', products = results)
```

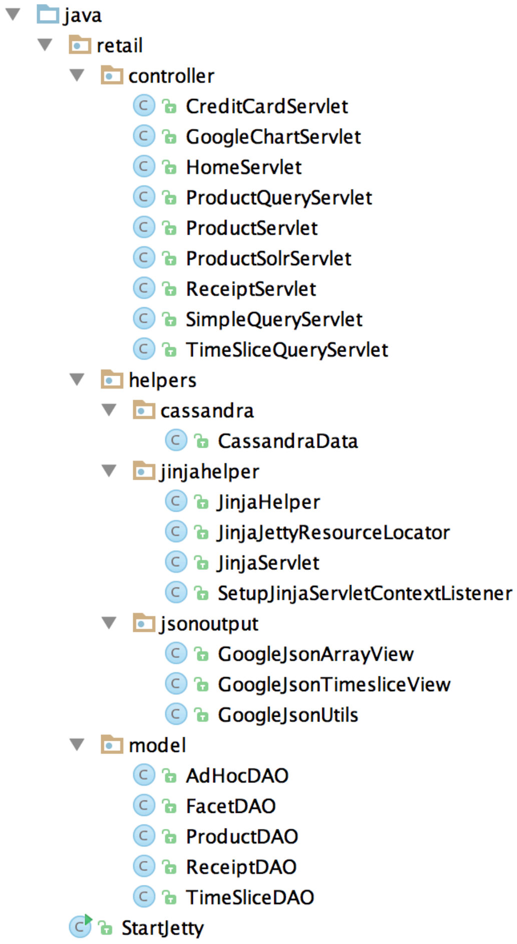
In the case of json, simply return the json string you build up:

```
thejson = dumps([description] + data, default=fix_json_format)
return thejson
```

The default=fix_json_format, call a method in rest.py to override the way certain datatypes are rendered. For example, dates are rendered as Date(m,d,y,h,m,s), a format used by google charts.

12 Java Implementation

12.1.1 Code Tree



```
▼ java
  ▼ retail
    ▼ controller
        CreditCardServlet
        GoogleChartServlet
        HomeServlet
        ProductQueryServlet
        ProductServlet
        ProductSolrServlet
        ReceiptServlet
        SimpleQueryServlet
        TimeSliceQueryServlet
    ▼ helpers
      ▼ cassandra
          CassandraData
      ▼ jinjahelper
          JinjaHelper
          JinjaJettyResourceLocator
          JinjaServlet
          SetupJinjaServletContextListener
      ▼ jsonoutput
          GoogleJsonArrayView
          GoogleJsonTimesliceView
          GoogleJsonUtils
    ▼ model
        AdHocDAO
        FacetDAO
        ProductDAO
        ReceiptDAO
        TimeSliceDAO
      StartJetty
```

12.1.2 Mainline

The mainline is StartJetty, and sets up the web application to handle the servlets and the static content. Note that the web.xml registers SetupJinjaServletContextListener to set up some jinja2 things, and hang it off of the ServletContext so it is accessible to the servlets.

12.1.3 Registering Routes

The Java implementation is using embedded Jetty with pure servlets to implement methods to service URL requests. In the retail sample application, there is generally a single servlet for each web page, but you may design servlets which serve several pages each. To register a class as a servlet, edit the webapp/META-INF/web.xml file. You register both the servlet, as well and the URL pattern or patterns that go with it.

```
<servlet>
   <servlet-name>ProductServlet</servlet-name>
   <servlet-class>retail.controller.ProductServlet</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>ProductServlet</servlet-name>
   <url-pattern>/web/product</url-pattern>
</servlet-mapping>
```

Implementing the servlet:

If your servlet is using the Jinja framework, it should extend JinjaServlet, otherwise it may extend HttpServelt. The Jinja servlet adds a render method which handles rendering the template with the correct encoding.

Retrieve the URL query parameters with the request

```
request.getParameter("product_id");
```

If you want a rest-style API, you need to parse out the URL string.

In the Java application, we have implemented DAO objects, so you should call them to interact with Cassandra. While a servlet can certainly make direct Cassandra calls, it runs counter to the MVC pattern.

```
ProductDAO product = ProductDAO.getProductById(product_id);
```

Then render the template. Place the template parameters in a Map<String,Object>, and call render. You need to write the rendered template to the servers output stream. This is necessary as the render function encodes the output in UTF-8 for browser compatibility.

```
Map<String, Object> context = Maps.newHashMap();
context.put("product", product);
byte[] renderedTemplate = render("/product_detail.jinja2", context);
out.write(renderedTemplate);
```

If the servlet return JSON

There are a number of helper functions in the jsonoutput to convert a ResultSet into a google JSON array. GoogleJSONArrayView works on regular rows. The special timeslice version expects each row to contain a map called quantities, and it will make each element of the map look like its own column.

12.2 Working With DAO Objects

Our DAO Objects in the model package are simply Java Beans that can be constructed from a cassandra Row object. In addition it has several static method to query the database and return a list of DAO objects. The DAO objects in this framework generally extend CassandraData, so they can call the method loadBeanFromRow to save code in copying the data from the cassandra row to the bean fields.

Alternatively, you can write your own constructor like the code below. loadbeanfromrow loops through all of the fields in the current row, and tries to find the setter function for that field. The setter must be of the form setSomeField for it to work.

```
productId = row.getString("product_id");
categoryId = row.getInt("category_id");
```

The DAO static methods will return a List<someDAO> or simply a someDAO if it is a singleton select. The special DAO objects AdHocDAO and TimeSliceDAO simply return ResultSets