TerraMeta Software, Inc.

CloudGraph Quick Start HBase (POJO)

1 Introduction

This step-by-step guide uses only annotated Java (POJO) objects as the source of schema or metadata. It shows how to build a Maven project which generates a simple HBase data model with 2 tables which inserts, queries and prints test data from HBase. It requires basic knowledge of the Java programing language, Apache Maven, HBase Server administration and assumes the following software install prerequisites.

- Java JDK 1.7 or Above
- Maven 3.x or Above
- HBase 1.0 or Above

See https://github.com/cloudgraph/cloudgraph-examples-quickstart for working examples which accomplany this guide.

2 CloudGraph Quick Start HBase (POJO)

2.1 Add Dependencies

Add the following dependency to your Maven project to get started.

```
<dependency>
      <groupId>org.terrameta</groupId>
      <artifactId>plasma-core</artifactId>
      <version>2.0.1</version>
</dependency>
<dependency>
      <groupId>org.cloudgraph</groupId>
      <artifactId>cloudgraph-hbase</artifactId>
      <version>1.0.8</version>
</dependency>
```

2.2 Create Entity POJOs

Next create a classic "Person-Org" data model using just Java POJO's. Create 4 Java enumeration classes annotated as below in a Java package called **examples.quickstart.hbase.pojo**. *(Note: Enumerations rather than Java classes are annotated to facilitate reuse across multiple code generation and metadata integration contexts. Your metadata is too valuable to relegate to a single context)*
The annotations capture typical structural metadata elements.

- Types and Aliases profiding logical / physical name isolation. See Type, Alias.
- Data Types. See DataProperty, ReferenceProperty.
- Cardinalities, Nullability and Visibility. See DataProperty, ReferenceProperty.
- Constraints. See ValueConstraint, EnumerationConstraint.
- Inheritance relationships (multiple inheritance is supported). See Type.
- Associations between entities. See ReferenceProperty.
- Enumerations as Domain Value Lists. See Enumeration.

In addition for HBase add the Table annotation and RowKeyField annotations in order to map specific entities to HBase tables and map specific entity fields to row key fields.

Enumeration 1 – OrgCat.java

```
package examples.quickstart.hbase.pojo;
import org.plasma.sdo.annotation.Alias;
import org.plasma.sdo.annotation.Enumeration;
```

```
@Enumeration(name = "OrgCat")
public enum OrgCat {
  @Alias(physicalName = "N")
  nonprofit,
  @Alias(physicalName = "G")
  government,
  @Alias(physicalName = "R")
  retail,
  @Alias(physicalName = "W")
  wholesale
}
```

Entity 1 – Party.java

```
@Type(name = "Party", isAbstract = true)
public enum Party {
  @Alias(physicalName = "CRTD_DT")
  @DataProperty(dataType = DataType.Date, isNullable = false)
  createdDate
}
```

Entity 2 – Person.java

```
package examples.quickstart.hbase.pojo;

import org.cloudgraph.store.mapping.annotation.RowKeyField;
import org.cloudgraph.store.mapping.annotation.Table;
import org.plasma.sdo.DataType;
import org.plasma.sdo.annotation.Alias;
import org.plasma.sdo.annotation.DataProperty;
import org.plasma.sdo.annotation.ReferenceProperty;
import org.plasma.sdo.annotation.Type;
import org.plasma.sdo.annotation.ValueConstraint;

@Table(name = "PERSON")
@Alias(physicalName = "PRS")
@Type(superTypes = { Party.class })
public enum Person {
  @RowKeyField
  @Key(type = KeyType.primary)
  @ValueConstraint(maxLength = "36")
  @Alias(physicalName = "FN")
  @DataProperty(dataType = DataType.String, isNullable = false)
  firstName,
  @RowKeyField
  @Key(type = KeyType.primary)
  @ValueConstraint(maxLength = "36")
  @Alias(physicalName = "LN")
  @DataProperty(dataType = DataType.String, isNullable = false)
  lastName,
  @ValueConstraint(totalDigits = "3")
  @Alias(physicalName = "AGE")
  @DataProperty(dataType = DataType.Int)
  age,
  @Alias(physicalName = "DOB")
  @DataProperty(dataType = DataType.Date)
  dateOfBirth,
  @Alias(physicalName = "EMP")
  @ReferenceProperty(targetClass = Organization.class, targetProperty =
"employee")
```

```
  employer;
}
```

Entity 3 – Organization.java

```
package examples.quickstart.hbase.pojo;

import org.cloudgraph.store.mapping.annotation.RowKeyField;
import org.cloudgraph.store.mapping.annotation.Table;
import org.plasma.sdo.DataType;
import org.plasma.sdo.annotation.Alias;
import org.plasma.sdo.annotation.DataProperty;
import org.plasma.sdo.annotation.EnumConstraint;
import org.plasma.sdo.annotation.ReferenceProperty;
import org.plasma.sdo.annotation.Type;
import org.plasma.sdo.annotation.ValueConstraint;

@Table(name = "ORGANIZATION")
@Alias(physicalName = "ORG")
@Type(superTypes = { Party.class })
public enum Organization {
  @RowKeyField
  @Key(type = KeyType.primary)
  @ValueConstraint(maxLength = "36")
  @Alias(physicalName = "NAME")
  @DataProperty(dataType = DataType.String, isNullable = false)
  name,
  @EnumConstraint(targetEnum = OrgCat.class)
  @Alias(physicalName = "ORG_CAT")
  @DataProperty(dataType = DataType.String, isNullable = false)
  category,
  @Alias(physicalName = "PARENT")
  @ReferenceProperty(isNullable = true, isMany = false, targetClass =
Organization.class, targetProperty = "child")
  parent,
  @Alias(physicalName = "CHILD")
  @ReferenceProperty(isNullable = true, isMany = true, targetClass =
Organization.class, targetProperty = "parent")
  child,
  @Alias(physicalName = "EMPLOYEE")
  @ReferenceProperty(isNullable = true, isMany = true, targetClass =
Person.class, targetProperty = "employer")
  employee;
}
```

2.3 Create Namespace POJO

In the same package as the above POJOs, create a file called package_info.java with the below annotates. These annotations associate the entities we created previously with a common namespace and data access context. For more information on applying annotations to package_into.java see https://www.intertech.com/Blog/whats-package-info-java-for

```
@Alias(physicalName = "HR")
@Namespace(uri = "http://plasma-quickstart-pojo/humanresources")
@NamespaceProvisioning(rootPackageName = "quickstart.pojo.model")
@NamespaceService(storeType = DataStoreType.NOSQL,
    providerName = DataAccessProviderName.HBASE,
properties = {
    "hbase.zookeeper.quorum=zookeeper-host1:2181,zookeeper-host2:2181",
    "hbase.zookeeper.property.clientPort=2181",
    "org.plasma.sdo.access.provider.hbase.ConnectionPoolMinSize=1",
    "org.plasma.sdo.access.provider.hbase.ConnectionPoolMaxSize=80" })
package examples.quickstart.pojo;
import org.plasma.runtime.annotation.NamespaceService;
import org.plasma.runtime.annotation.NamespaceProvisioning;
import org.plasma.sdo.annotation.Namespace;
import org.plasma.sdo.annotation.Alias;
import org.plasma.runtime.DataAccessProviderName;
import org.plasma.runtime.DataStoreType;
```

Namespace 1 – package_info.java

2.4 Add Plasma Maven Plugin

Add the CloudGraph Maven Plugin with 2 executions which generate data access and query (DSL) classes. See below CloudGraph Maven Plugin Configuration for complete listing.
See https://github.com/cloudgraph/cloudgraph-examples-quickstart for working examples which accomplany this guide.

2.5 Generate Source

After adding the plugin and 2 executions type:

```
    maven generate-sources
```

The generated data access source code should appear under `target/generated-sources/quickstart.pojo.model` which is the package we specified in @NamespaceProvisioning on the namespace.

2.6 Add Run Time Dependencies

Next, add the following additional dependencies to your Maven project, including an HBase data access service provider (CloudGraph HBase).

```
<dependency>
    <groupId>org.cloudgraph</groupId>
    <artifactId>cloudgraph-hbase</artifactId>
    <version>1.0.8</version>
</dependency>
```

2.7 Insert and Query HBase Data

And finally create a class as below which inserts 2 organizations (parent and child) with a single employee under the child. The example then queries for the “graph” traversing the foreign key refrences from the person (as a root) back to the employer organization and then the parent organization. Then final y the example prints the serialized result graph as formatted XML for easy visualization and debugging. The final output should look like the below XML example. See https://github.com/cloudgraph/cloudgraph-examples-quickstart for working examples which accomplany this guide.

Figure 3 – Result Graph, Serialized as XML

```
<ns1:Person xmlns:ns1="http://plasma-quickstart-pojo/humanresources"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
  firstName="Mark" lastName="Hamburg (097161)" age="55" createdDate="2017-10-
06T07:00:00">
  <employer name="Best Buy Sales (097161)">
    <parent name="Best Buy Corporation Inc. (097161)" category="R"></parent>
  </employer>
</ns1:Person>
```

Figure 4 – Inser/Query HBase Data

```
package examples.quickstart;

import java.io.IOException;
import java.util.Date;
import org.plasma.runtime.*;
import org.plasma.sdo.*;
import org.plasma.sdo.access.client.*;
import org.plasma.sdo.helper.*;
import quickstart.pojo.model.*;
import quickstart.pojo.model.query.QPerson;
import commonj.sdo.*;

public class ExampleRunner {

  public static CloudGraphDataGraph runExample() throws IOException {
    SDODataAccessClient client = new SDODataAccessClient(
        new PojoDataAccessClient(DataAccessProviderName.HBASE));

    DataGraph dataGraph = CloudGraphDataFactory.INSTANCE.createDataGraph();
    dataGraph.getChangeSummary().beginLogging();
    Type rootType = CloudGraphTypeHelper.INSTANCE.getType(Organization.class);
    String randomSuffix = String.valueOf(System.nanoTime()).substring(10);

    Organization org = (Organization) dataGraph.createRootObject(rootType);
    org.setName("Best Buy Corporation Inc. (" + randomSuffix + ")");
    org.setCategory(OrgCat.RETAIL.getInstanceName());
    org.setCreatedDate(new Date());

    Organization child = org.createChild();
    child.setName("Best Buy Sales (" + randomSuffix + ")");
    child.setCategory(OrgCat.RETAIL.getInstanceName());
    child.setCreatedDate(new Date());

    Person pers = child.createEmployee();
    pers.setFirstName("Mark");
    pers.setLastName("Hamburg (" + randomSuffix + ")");
    pers.setAge(55);
    pers.setCreatedDate(new Date());

    client.commit(dataGraph, ExampleRunner.class.getSimpleName());

    QPerson query = QPerson.newQuery();
    query.select(query.wildcard())
         .select(query.employer().name())
         .select(query.employer().parent().name())
         .select(query.employer().parent().category());
    query.where(query.firstName().eq("Mark")
         .and(query.lastName().like("Ham*")));
    DataGraph[] results = client.find(query);
    return (CloudGraphDataGraph) results[0];
  }

  public static void main(String[] args) {
    try {
      CloudGraphDataGraph graph = runExample();
      System.out.println(graph.asXml());
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
```

See https://github.com/cloudgraph/cloudgraph-examples-quickstart for working examples which accomplany this guide.

3 CloudGraph Maven Plugin Configuration

Below is the Maven plugin listing referenced about which is needed for generation of data access source code and DDL. See https://github.com/cloudgraph/cloudgraph-examples-quickstart for working examples which accomplany this guide.

```
<plugin>
      <groupId>org.terrameta</groupId>
      <artifactId>plasma-maven-plugin</artifactId>
      <version>2.0.0</version>
      <dependencies>
            <dependency>
                  <groupId>org.terrameta</groupId>
                  <artifactId>plasma-core</artifactId>
                  <version>2.0.1</version>
            </dependency>
            <dependency>
                  <groupId>org.cloudgraph</groupId>
                  <artifactId>cloudgraph-hbase</artifactId>
                  <version>1.0.8</version>
            </dependency>
      </dependencies>
      <executions>
            <execution>
                  <id>sdo-create</id>
                  <configuration>
                        <action>create</action>
                        <dialect>java</dialect>
                        <additionalClasspathElements>
                              <param>${basedir}/target/classes</param>
                        </additionalClasspathElements>
                        <outputDirectory>${basedir}/target/generated-
sources/java</outputDirectory>
                  </configuration>
                  <goals>
                        <goal>sdo</goal>
                  </goals>
            </execution>
            <execution>
                  <id>dsl-create</id>
                  <configuration>
                        <action>create</action>
                        <dialect>java</dialect>
                        <additionalClasspathElements>
                              <param>${basedir}/target/classes</param>
                        </additionalClasspathElements>
                        <outputDirectory>${basedir}/target/generated-
sources/java</outputDirectory>
                  </configuration>
                  <goals>
                        <goal>dsl</goal>
                  </goals>
            </execution>
      </executions>
</plugin>
.
```

4 Maven Compiler Plugin Configuration

We use 2 executions in the compiler plugin because the annotation discovery for your annotated Java requires COMPILED classes. The compiled annotated classes are first used at generate-sources phase, then for several later Maven phases. An alternative to this "trick" is to isolated your annotated classes in a separate compiled Maven module, then perform the code generation in a second module which depends on the first. See https://github.com/cloudgraph/cloudgraph-examples-quickstart for working examples which accomplany this guide.

```
<plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration>
            <source>1.7</source>
            <target>1.7</target>
            <encoding>UTF-8</encoding>
      </configuration>
      <executions>
            <execution>
                  <id>default-compile</id>
                  <phase>generate-sources</phase>
                  <configuration>
                        <excludes>
                              <exclude>**/generated-sources/*</exclude>
                              <exclude>**/examples/quickstart/*</exclude>
                        </excludes>
                  </configuration>
            </execution>
            <execution>
                  <id>compile-generated</id>
                  <phase>compile</phase>
                  <goals>
                        <goal>compile</goal>
                  </goals>
                  <configuration>
                  </configuration>
            </execution>
        </executions>
    </plugin>
```