Google Data Store

Rakesh Vidyadharan
rakesh@sptci.com

2010-05-18

Google Data Store

- A distributed key-value data store built on BigTable.
- Most google projects including search engine indices, Google Earch data etc. are stored in BigTable.
- The GDS may be thought of as a “sparse, distributed, persistent, multi-demensional sorted map”.
- Standard JDO/JPA API and low level API.
- Local version available for unit testing.
- Pay Per Use available.

Data Types

- String.
- All numeric primitive wrappers (not java.math.BigInteger or java.math.BigDecimal).
- Key, for storing references to other Entity objects.
- User, for storing references to users.
- ShortBlob, for storing binary data small enough to be indexed.
- Blob, for storing unindexed binary data less than 1MB.
- Text, for storing unindexed String data less than 1MB.
- BlobKey, for storing references to user uploaded blobs to the BlobStore (up to 1Gb).
- Date.
- Link.
- Collections of any of the above.

Entity Groups

- A group defines types of entities that can be manipulated within the same transaction.
- All entities within an entity group reside in the same datastore node.
- An entity without a parent is the root entity of an entity group.
- The chain of parent entities up to the root represents the path of an entity in the datastore.
- Owned relationships in JDO/JPA must fall within the same entity group.

Query Cursors

- Retrieve subsequent results of a query.
  More efficient than using offsets.
- A base64 encoded string that represents the location of the last fetched result.
- The encoded string contains information about the GAE application id, entity kind, and key. May not be safe to send to a client.
- New entities added that fall before the location of the cursor will not be seen in a subsequent query.

GDS Benefits

- Excellent read performance.
- Excellent query performance.
- Transparent redundant storage and load balancing.
- Flexible data structure.
- Query indexes.
- Accessible from multiple apps hosted on GAE.

GDS Limitations

- Slow writes.
- Result set restricted to 1000 items.*
- Searchable text is limited to 500 characters.
- Maximum size of a property is 1Mb.
- Cannot execute arbitrary queries.
- Maximum number of entities in a batch save/delete is 500.
- Number of indexes limited to 100 (200 for paid) per database.
- Database size limited to 1Gb (free).
- Limitations of number of queries executed, data sent or received from API, CPU time etc.

Query Limitations

- No join queries.
- Cannot filter by `null`.
- Inequality filters are allowed only on one property.
- Properties in inequality filters must be sorted before other sort orders
- Inequality filters on multi-value properties will return results only if one of the values uniquely matches the filter.

JDO/JPA Access

- Model your objects as POJO's and annotate them using JDO/JPA interfaces.
- Persist and access the POJO's using JDO-QL or JPA-QL.
- Complex object graphs not supported at present.
- Related object key fields need to inherit parent key.
- Fields of same class not supported.

Sample JDO Code

```
import static
  javax.jdo.JDOHelper.getPersistenceManagerFactory;
import javax.jdo.PersistenceManager;
import javax.jdo.PersistenceManagerFactory;

PersistenceManagerFactory pmf =
  getPersistenceManagerFactory(
      "transactions-optional");
PersistenceManager pm =
  pmf.getPersistenceManager();
```

```
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;

@PersistenceCapable
public class User
{
  @PrimaryKey
  private String email;
  @Persistent
  private String name;
  @Persistent
  private String description;
}
```

```
final User user = new User();
user.setEmail( "user@test.com" );
user.setName( "New User" );
user.setDescription( "Testing new user" );

try
{
  pm.makePersistent( user );
}
finally
{
  pm.close();
}
```

```
final String email = "user@test.com";

try
{
  User user = pm.getObjectById( User.class, email )
  user.setDescription( "Updated user description" )
  pm.makePersistent( user );
}
finally
{
  pm.close();
}
```

```
import javax.jdo.Query;

final String name = "New User";
final Query query = pm.newQuery( User.class );
query.setFilter( "this.name == :name" );

try
{
  Collection<User> users =
    (Collection<User>) query.execute( name );
  for ( final User user : users ) { /* logic */ }
}
finally
{
  query.closeAll();
}
```

Low-Level API Access

- Model your data as Entity instances.
- Entities are similar in concept and structure to a Map or JCR Node.
- Entity properties are automatically indexed.
- All entities have a GDS specific object identity - Key. The key may be converted back and forth between a "web-safe" string.

Imports

```
import static com.google.appengine.api.datastore.
    DatastoreServiceFactory.getDatastoreService;
import com.google.appengine.api.datastore.
    DatastoreService;
import com.google.appengine.api.datastore.Entity;
import com.google.appengine.api.datastore.Key;
import static com.google.appengine.api.datastore.
    KeyFactory.keyToString;
import static com.google.appengine.api.datastore.
    KeyFactory.stringToKey;
```

Imports

```
import com.google.appengine.api.datastore.
    PreparedQuery;
import com.google.appengine.api.datastore.Query;
import com.google.appengine.api.datastore.
    Query.FilterOperator;
import com.google.appengine.api.datastore.Text;
```

Create an Entity

```
final DatastoreService pm = getDatastoreService();
final String type = "User";
final Entity entity = new Entity( type );
entity.setProperty( "email", "user@test.com" );
entity.setProperty( "name", "New User" );
entity.setProperty( "description",
    "Testing new user" );
final Key key = pm.put( entity );
return keyToString( key );
```

Query for Entities

```
final String name = "New User";
final Query query = new Query( type );
query.addFilter(
    "name", FilterOperator.EQUAL, name );

final PreparedQuery pquery = pm.prepare( query );

for ( final Entity entity : pquery.asIterable() )
{
  // process entity
}

//final Entity entity = pquery.asSingleEntity();
//pquery.asList( withLimit( 10 ).offset( 5 ) );
```

Lookup Entity

```
final Key key = stringToKey( keyString );
final Entity entity = pm.get( key );
Set<String> properties =
  entity.getProperties().keySet();

if ( ! properties.contains( "another" ) )
{
  entity.setProperty(
      "another", "A new property" );
}

pm.put( entity );
```

Datastore Statistics

```
final Query query = new Query( "__Stat_Total__" );
final PreparedQuery pquery = pm.prepare( query );
final Entity stats = pquery.asSingleEntity();
final long totalBytes =
  (Long) stats.getProperty( "bytes" );
final long totalEntities =
  (Long) stats.getProperty( "count" );
```

Query Indexes

- Queries using a single predicate do not require special handling.
- Queries with multiple conditions need to have a composite index that covers all the predicates.
- The index definitions must be included in your application as `/META-INF/datastore-indexes.xml`.
- Indexes may be autogenerated while unit testing.