

## HIBERNATE SHARD

Hien Nguyen

MIT OCW

### **Introduction**

Distributed system has enabled the scalability to distribute data across multiple databases which transact against large physical distributed datasets. Hibernate shard is one option for horizontal data partitioning which built on top of Hibernate, encapsulate and reduce the complexity of building applications that work with sharded datasets. Partitioning can improve scalability, high availability, reduce contention and optimize performance. There are 3 typical partition strategies: horizontal, vertical & functional partitioning. This report paper is only for horizontal sharding. The most important factor is the choice of sharding key that ensure data is partitioned to spread the workload as evenly as possible across the shards. Also, it is more important to balance number of requests

Hibernate shard is a project that facilitates working with database architecture that are sharded by providing encapsulation to the horizontal dataset.

HS is a framework that designed to minimize the complexity by adding support for horizontal partitioning on top of Hibernate Core. It was created by a group of Google Engineers, more documentation about HS architecture, concepts and design and can read on HS site.

#### Key features

- Standard hibernate programming model
- Flexible sharding algorithms
- Support virtual shards

Hibernate Shard has 2 main ideas: generalized sharding logic & application specific sharding logic. Shard strategy provides ShardAccessStrategy which are SequentialShardAccessStrategy & ParallelShardAccessStrategy.

Sharding Algorithms: Round robin implementation is provided to get started

ID generation

Virtual shards supports the capacity of redistribute the data across the shards either to achieve proper load balancing algorithms, for example.

## Methods

### **Hibernate shards maven dependency**

Hibernate Shard is distribute from Sourceforge version 3.0.0.Beta2

<http://sourceforge.net/projects/hibernate/files/hibernate-shards/3.0.0.Beta2/>

## Requirements

System requirements: Hibernate Core, Java 1.5+

We will create a research application (not production level code) to experience the use of Hibernate Shard.

**Entity.** NationalPlayer indicates the shard databases

```
@Entity
@Table (name="NATIONAL_PLAYER")
public class NationalPlayer {
    @Id @GeneratedValue(generator="PlayerIdGenerator")
    @GenericGenerator(name="PlayerIdGenerator", strategy="org.hibernate.shards.id.ShardedUUIDGenerator")
    @Column(name="PLAYER_ID")
    private BigInteger id;

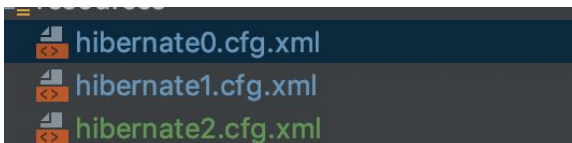
    @Column (name="FIRST_NAME")
    private String firstName;

    @Column (name="LAST_NAME")
    private String lastName;

    @Column (name="CAREER_GOALS")
    private int careerGoals;

    @Column (name="WORLD_RANKING")
    private int worldRanking;
```

**Hibernate config.** 3 sharded databases is defined using hibernate config file.



```
hibernate0.cfg.xml
hibernate1.cfg.xml
hibernate2.cfg.xml
```

```

<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory name="HibernateSessionFactory0">
    <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
    <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
    <property name="connection.url">jdbc:hsqldb:mem:shard0</property>
    <property name="connection.username">sa</property>
    <property name="connection.password"></property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <property name="hibernate.connection.shard_id">0</property>
    <property name="hibernate.shard.enable_cross_shard_relationship_checks"> true </property>
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.format_sql">true</property>
    <property name="hibernate.jdbc.batch_size">20</property>
  </session-factory>
</hibernate-configuration>

```

**Shard Strategy.** In order to route data to appropriate, we define a custom shard selection strategy that use country code of Player to route persistence to a particular databases

```

private static ShardStrategyFactory buildShardStrategyFactory() {
    ShardStrategyFactory shardStrategyFactory = (shardIds) -> {
        RoundRobinShardLoadBalancer loadBalancer = new RoundRobinShardLoadBalancer(shardIds);
        ShardSelectionStrategy pss = new org.ut.biolab.ShardSelectionStrategy(loadBalancer);
        ShardResolutionStrategy prs = new AllShardsShardResolutionStrategy(shardIds);
        ShardAccessStrategy pas = new SequentialShardAccessStrategy();

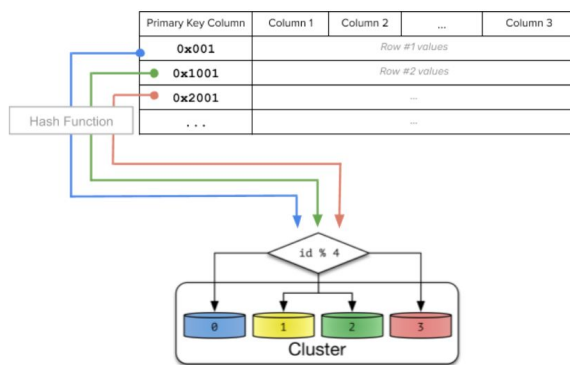
        return new ShardStrategyImpl(pss, prs, pas);
    };
    return shardStrategyFactory;
}

```

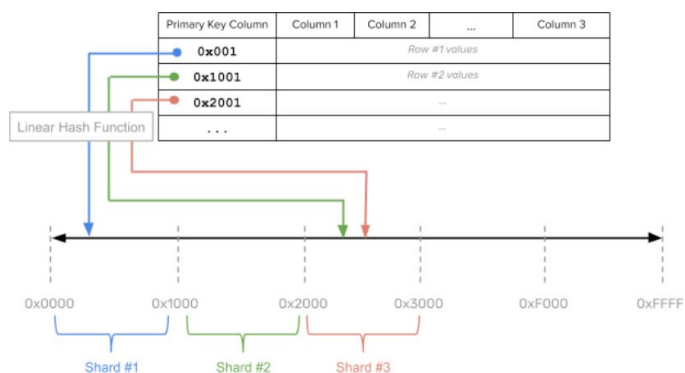
## Sharding algorithm

We are also interested in investigate other distribution storage algorithmic sharding strategies.

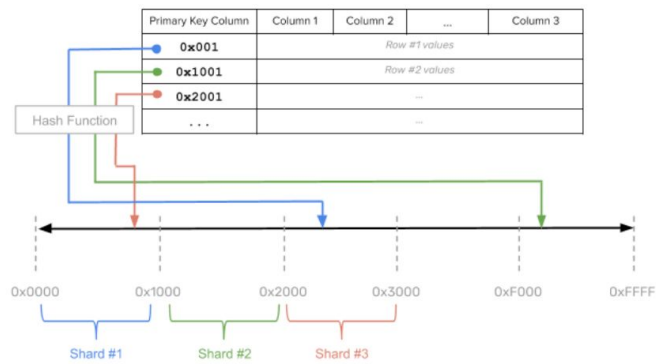
- Memcached & redis common used technique is compute a numeric hash value out of the key and computing modulo of that hash using total number of nodes to compute which node owns the key



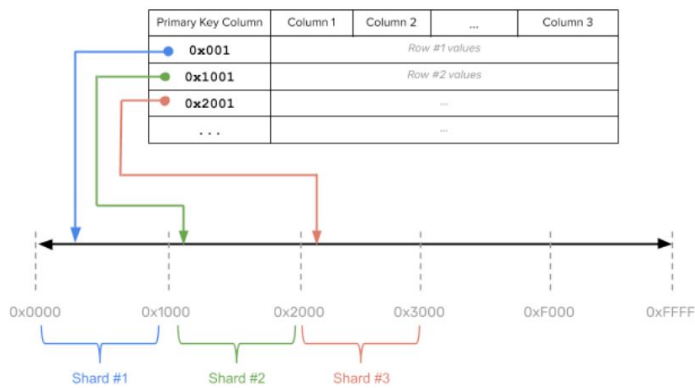
- Cassandra implements linear hash sharding, known as OrderPreservingPartitioner. Linear hash sharding is a hybrid between hash and range sharding that preserve the sort order of the rows by using linear hash functions instead of random hash function to compute how to shard the rows.



- DynamoDB uses consistent hash sharding which data is evenly and randomly distributed across shards using a partitioning algorithm. Each row of table is placed into a shard determined by computing a consistent hash on the partition column values of that row



- Apache HBase supports range sharding involves splitting the rows of a table into contiguous ranges that respect the sort order of the table based on primary key column values.



## Results

Unit tests are implemented:

### Unit test 1

When players are persisted, they are stored in appropriate database instance

```

@Test
public void testShardingPersistence() {
    BigInteger indiaPlayerId = null;
    BigInteger usaPlayerId = null;
    BigInteger italyPlayerId = null;

    // Save all three players
    savePlayer(indiaPlayer);
    savePlayer(usaPlayer);
    savePlayer(italyPlayer);

    indiaPlayerId = indiaPlayer.getId();
    System.out.println("Indian Player Id:" + indiaPlayerId);

    usaPlayerId = usaPlayer.getId();
    System.out.println("Usa Player Id:" + usaPlayerId);

    italyPlayerId = italyPlayer.getId();
    System.out.println("Italy Player Id:" + italyPlayerId);

    assertNotNull( message: "Indian Player must have been persisted", getShardPlayer(indiaPlayerId));
    assertNotNull( message: "Usa Player must have been persisted", getShardPlayer(usaPlayerId));
    assertNotNull( message: "Italy Player must have been persisted", getShardPlayer(italyPlayerId));

    // Ensure that the appropriate shards contain the players
    assertExistsOnlyOnShard( message: "Indian Player should have existed on only shard 0", shardId: 0, indiaPlayerId);
    assertExistsOnlyOnShard( message: "Usa Player should have existed only on shard 1", shardId: 1, usaPlayerId);
    assertExistsOnlyOnShard( message: "Italian Player should have existed only on shard 2", shardId: 2, italyPlayerId);
}

```

## Unit test 2

Query executed against the shard will obtain data from all the sharded databases accurately.

```

@Test
public void testSimpleCriteria() throws Exception {
    savePlayer(indiaPlayer);
    savePlayer(usaPlayer);
    savePlayer(italyPlayer);

    Session session = HibernateUtil.getSession();
    Transaction tx = session.beginTransaction();

    try {
        Criteria c = session.createCriteria(NationalPlayer.class)
            .add(Restrictions.eq("country", Country.INDIA));
        List<NationalPlayer> players = c.list();
        assertTrue("Should only return the sole india Player", condition: players.size() == 1);
        assertContainsPlayers(players, indiaPlayer);

        c = session.createCriteria(NationalPlayer.class)
            .add(Restrictions.gt("careerGoals", value: 50));
        players = c.list();
        assertEquals("Should return the usa and india players", expected: 2, players.size());
        assertContainsPlayers(players, indiaPlayer, usaPlayer);

        c = session.createCriteria(NationalPlayer.class)
            .add(Restrictions.between("worldRanking", lo: 5, hi: 15));
        players = c.list();
        assertEquals("Should only have the usa player", expected: 1, players.size());
        assertContainsPlayers(players, usaPlayer);

        c = session.createCriteria(NationalPlayer.class)
            .add(Restrictions.eq("lastName", value: "Acharya"));
        players = c.list();
        assertEquals("All Players should be found as they have same last name", expected: 3, players.size());
        assertContainsPlayers(players, indiaPlayer, usaPlayer, italyPlayer);
        tx.commit();
    } catch (Exception e) {
        tx.rollback();
    }
}

```

## Further research & development

Unbalance partition when you have number of busy process in 1 database but many several idle.

Hibernate Shard has indicated several limitations that might need some attention for research such as cross shard object graph, distributed transaction, stateful interceptors or replicated data. (Refer to Hibernate Shard documentation for more information).

Partition key



## Reference

Hibernate source code: <https://github.com/hibernate/hibernate-shards>

<https://sleeplessinslc.blogspot.com/2008/09/hibernate-shards-maven-simple-example.html>

<https://docs.jboss.org/hibernate/stable/shards/reference/en/html>

<https://www.ibm.com/developerworks/library/j-javadev2-11/>

Sharding by hash partitioning- A database scalability pattern to achieve evenly

sharded database clusters paper by CaioHCosta.

[https://pdfs.semanticscholar.org/db7b/23b1847671b2bfbcf032cd67533520fbf313.pdf?\\_ga=2.175802274.1085564224.1601629015-1074301603.1601629015](https://pdfs.semanticscholar.org/db7b/23b1847671b2bfbcf032cd67533520fbf313.pdf?_ga=2.175802274.1085564224.1601629015-1074301603.1601629015)

Amazon sharding database white paper

Microsoft sharding database documentation