Hibernate - Batch Processing

Last Updated : 23 Jul, 2025

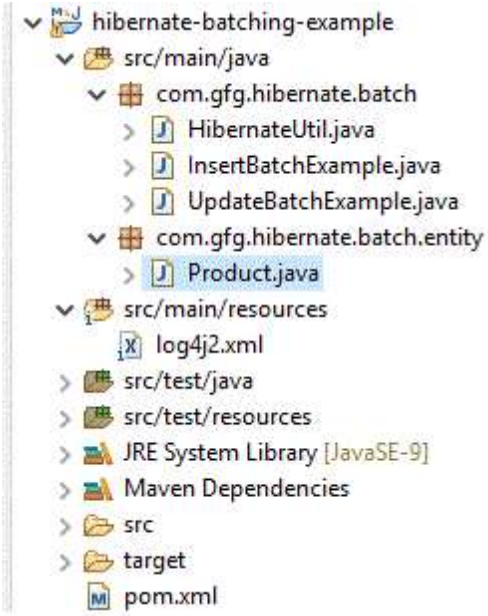
Hibernate is storing the freshly inserted objects in the second-level cache. Because of this, there is always the possibility of OutOfMemoryException when Inserting more than one million objects. But there will be situations to inserting huge data into the database. This can be accomplished by batching in hibernate.

> ***Note**: hibernate.jdbc.batch_size has to be set and it is an integer between 10 and 50. If zero or negative value is set, then it disables batching.*

Let us use the JPA @TableGenerator annotation to generate the unique key for the entity. As we are not sure of how many records were inserted because of the batch, the IDENTITY generator is disabled by Hibernate.

Example Project

Project Structure:



This is a maven-driven project

pom.xml

```
<project xmlns="https://maven.apache.org/POM/4.0.0"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="https://maven.apache.org/POM/4.0.0
                       https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.gfg.hibernate.batch</groupId>
   <artifactId>hibernate-batching-example</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <properties>
      <maven.compiler.source>1.8</maven.compiler.source>
      <maven.compiler.target>1.8</maven.compiler.target>
   </properties>
<dependencies>
   <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>5.4.15.Final</version>
   </dependency>
   <!-- mysql connector dependency -->
   <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.34</version>
   </dependency>
   <dependency>
      <groupId>javax.xml.bind</groupId>
      <artifactId>jaxb-api</artifactId>
      <version>2.3.0</version>
   </dependency>

   <!--Log4j2 API -->
```

```
    <dependency>
       <groupId>org.apache.logging.log4j</groupId>
       <artifactId>log4j-core</artifactId>
       <version>2.11.0</version>
    </dependency>
    <dependency>
       <groupId>org.apache.logging.log4j</groupId>
       <artifactId>log4j-api</artifactId>
       <version>2.11.0</version>
    </dependency>
</dependencies>
</project>
```

For logging purposes, we are using **log4j2.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
   <Appenders>
      <!-- Console Appender -->
      <Console name="Console" target="SYSTEM_OUT">
         <PatternLayout pattern="%d{yyyy-MMM-dd HH:mm:ss a} [%t] %-5level
%logger{36} - %msg%n" />
      </Console>
   </Appenders>
   <Loggers>
      <!-- Log everything in hibernate -->
      <Logger name="org.hibernate.engine.jdbc.batch.internal.BatchingBatch"
level="debug" additivity="false">
         <AppenderRef ref="Console" />
      </Logger>
      <Root level="error">
         <AppenderRef ref="Console" />
      </Root>
   </Loggers>
</Configuration>
```

Let us see the important files of the application. Let's start with the entity class

Product.java

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.TableGenerator;

@Entity
public class Product {

    @Id
    @TableGenerator(name = "PRODUCT_SEQ")
```

```
    // As we are doing batch insert, using TableGenerator,
    // unique key is determined
    @GeneratedValue(strategy = GenerationType.TABLE,
                    generator = "PRODUCT_SEQ")
    // data members of product
    private Long id;

    private String productName;
    private String productBrand;
    private int price;
    // Getter and setters

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public String getProductName() { return productName; }

    public void setProductName(String productName)
    {
        this.productName = productName;
    }

    public String getProductBrand() { return productBrand; }

    public void setProductBrand(String productBrand)
    {
        this.productBrand = productBrand;
    }

    public int getPrice() { return price; }

    public void setPrice(int price) { this.price = price; }
    // This is essentially required to avoid exceptions
    public Product() {}
}
```

Let us see the main Util class

HibernateUtil.java

```
import com.gfg.hibernate.batch.entity.Product;
import java.util.HashMap;
import java.util.Map;
import org.hibernate.SessionFactory;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Environment;

public class HibernateUtil {
    private static StandardServiceRegistry registry;
    private static SessionFactory sessionFactory;
```

```
    public static SessionFactory getSessionFactory()
    {
        if (sessionFactory == null) {
            try {
                StandardServiceRegistryBuilder
                    registryBuilder
                    = new StandardServiceRegistryBuilder();

                // Configuration properties
                Map<String, Object> settings
                    = new HashMap<>();

                settings.put(Environment.DRIVER,
                             "com.mysql.jdbc.Driver");

                settings.put(
                    Environment.URL,
                    "jdbc:mysql://localhost:3306/geeksforgeeks?
serverTimezone=UTC");
                // Specify mySQL credentials here
                settings.put(Environment.USER, "root");
                settings.put(Environment.PASS, "admin");

                settings.put(Environment.HBM2DDL_AUTO,
                             "update");
                // Set JDBC batch size. It can be set
                // between 10 and 50
                settings.put(
                    Environment.STATEMENT_BATCH_SIZE, 50);

                registryBuilder.applySettings(settings);
                registry = registryBuilder.build();

                MetadataSources sources
                    = new MetadataSources(registry);

                // This entity class Product is going to be
                // used for batch insert or update
                sources.addAnnotatedClass(Product.class);
                Metadata metadata
                    = sources.getMetadataBuilder().build();

                sessionFactory
                    = metadata.getSessionFactoryBuilder()
                          .build();
            }
            catch (Exception e) {
                if (registry != null) {
                    StandardServiceRegistryBuilder.destroy(
                        registry);
                }
                e.printStackTrace();
            }
        }
        return sessionFactory;
```

```
    }

    public static void shutdown()
    {
        if (registry != null) {
            StandardServiceRegistryBuilder.destroy(
                registry);
        }
    }
}
```

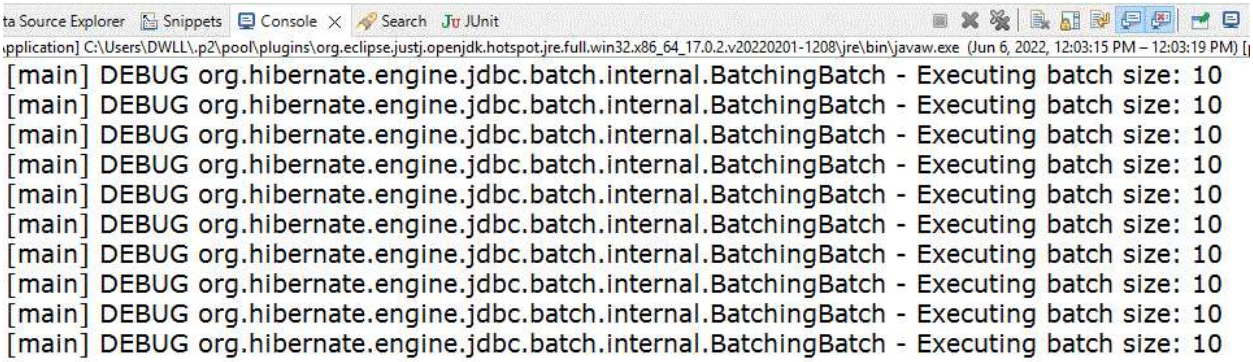
Let us see how to batch insert can be happened via

InsertProductBatchExample.java

```
import com.gfg.hibernate.batch.entity.Product;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class InsertProductBatchExample {
    public static void main(String[] args)
    {
        Session session = null;
        Transaction transaction = null;
        // Setting zero or negative number will disable the
        // batching.
        int batchSize
            = 10; // As of now, it is hardcoded to 10
        try {
            session = HibernateUtil.getSessionFactory()
                          .openSession();
            transaction = session.beginTransaction();
            // Here as a sample 100 items are inserted, but
            // it can be changed as per user choice
            for (long idx = 1; idx <= 100; idx++) {
                Product product = new Product();
                // We can use this as sample. Please change
                // according to the requirement
                product.setProductName("Product" + idx);
                product.setProductBrand("A");
                product.setPrice((int)idx * 10);
                session.save(product);
                if (idx > 0
                    && idx % batchSize
                           == 0) { // Keep on doing this
                                   // step in order to
                                   // continue and avoid
                                   // exceptions
                    session.flush();
                    session.clear();
                }
            }
            transaction.commit();
        }
```

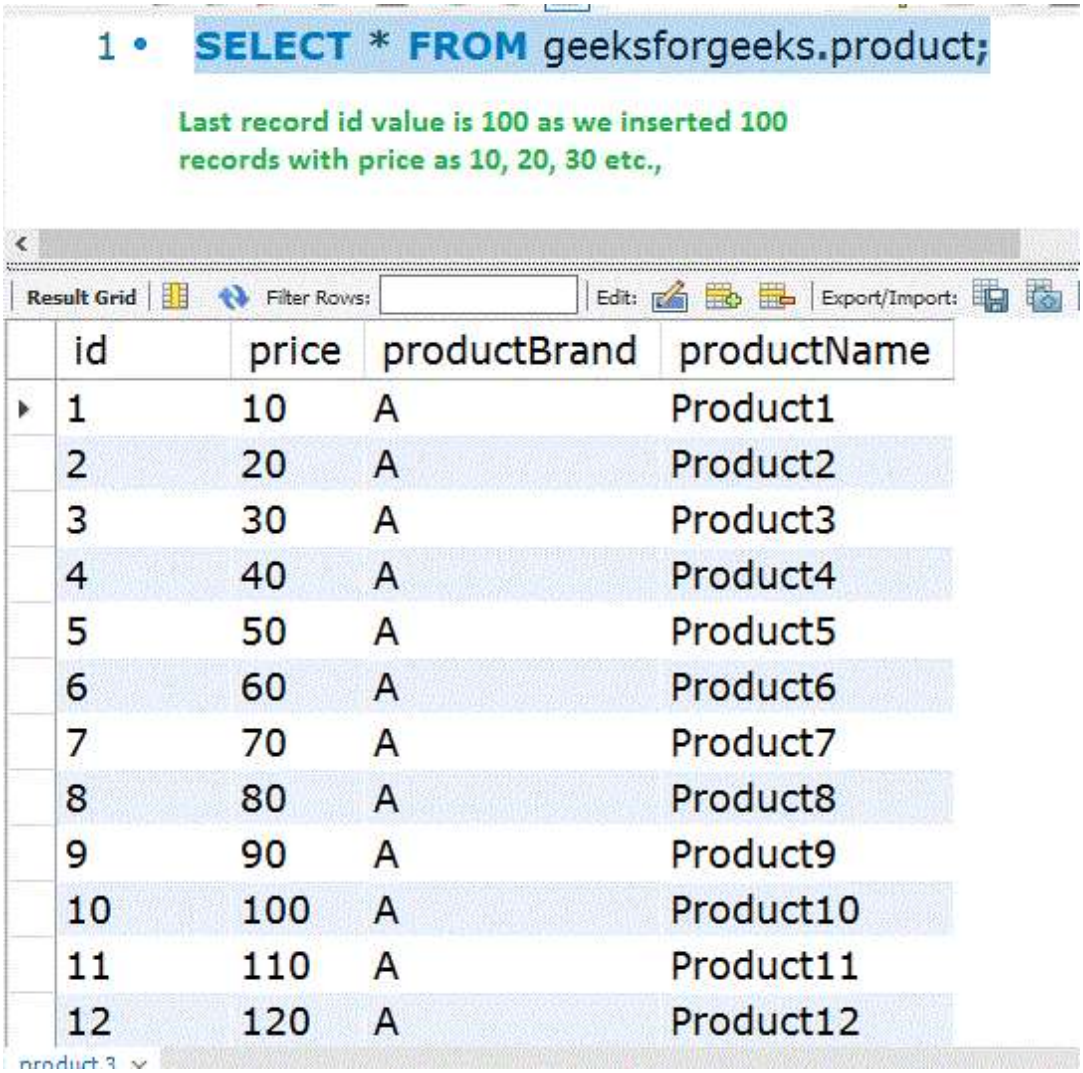
```
        catch (Exception e) {
            e.printStackTrace();
        }
        finally {
            if (session != null) {
                session.close();
            }
        }

        HibernateUtil.shutdown();
    }
}
```

On execution of the above program, in the console, we can able to see as



We are seeing 10 times, because we have given batch size as 10 and tried to insert by running the for loop 100 times. So 100/10 = 10 times we are seeing this output.

Let us check the same with MySQL output as well

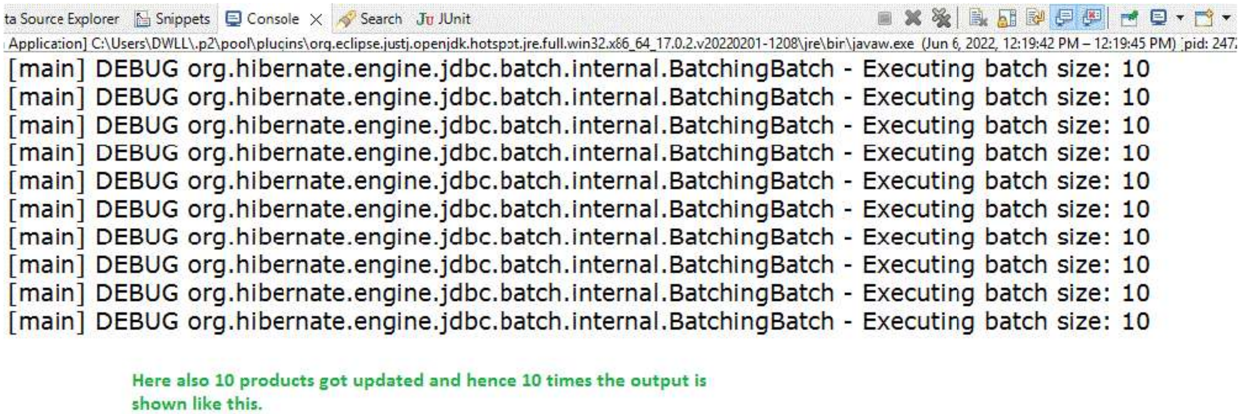


Now let us see the same for updates as well.

UpdateProductBatchExample.java

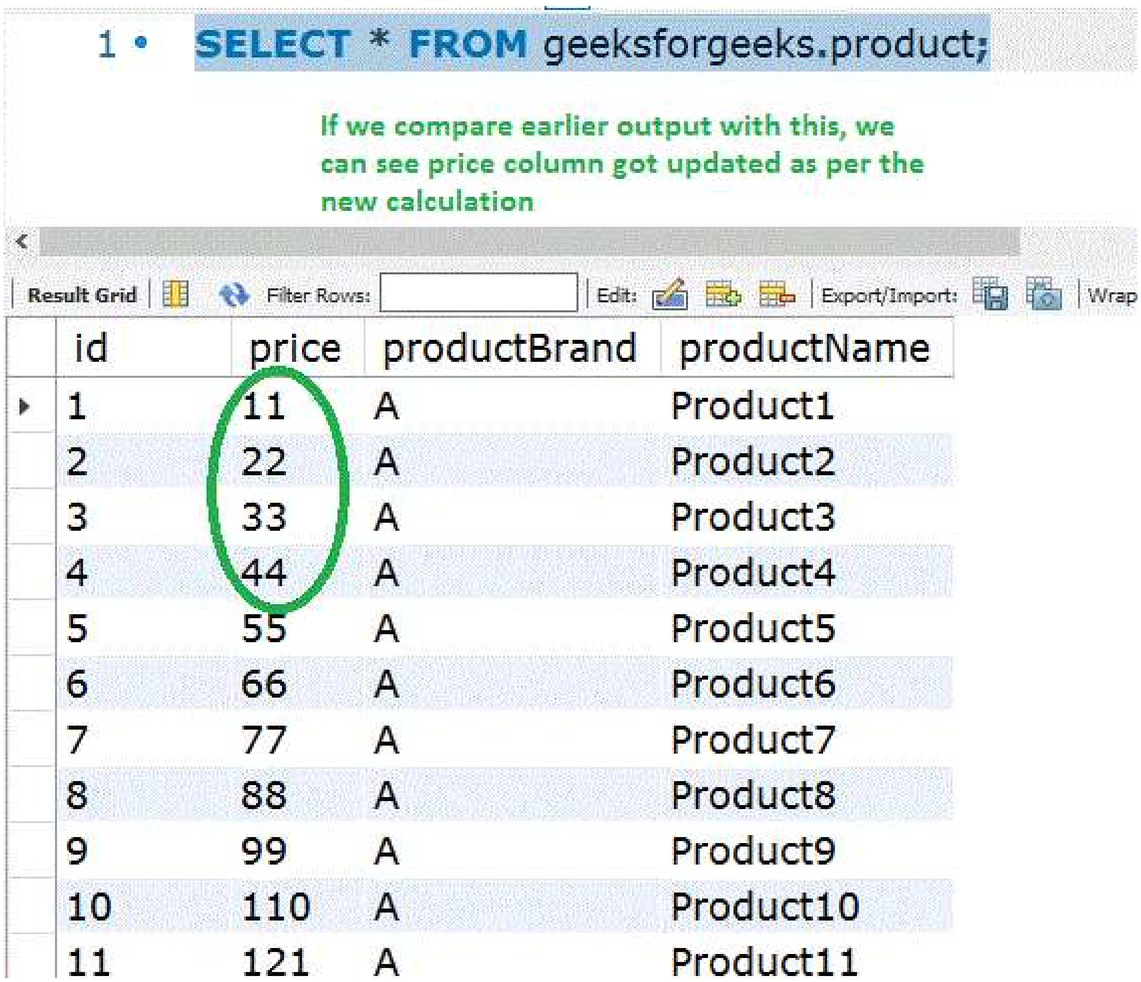
```
import com.gfg.hibernate.batch.entity.Product;
import org.hibernate.CacheMode;
import org.hibernate.ScrollMode;
import org.hibernate.ScrollableResults;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class UpdateProductBatchExample {
    public static void main(String[] args)
    {
        Session session = null;
        Transaction transaction = null;
        // As we are going to do bulk operations, we need
        // ScrollableResults
        ScrollableResults scrollableResults = null;
        // Setting zero or negative number will disable the
        // batching.
        int batchSize = 10; // It can be between 10 and 50.
```

```
        try {
            session = HibernateUtil.getSessionFactory()
                          .openSession();
            transaction = session.beginTransaction();
            scrollableResults
                = session
                      .createQuery(
                          "from Product") // Query the table
                      .setCacheMode(CacheMode.IGNORE)
                      .scroll(
                          ScrollMode
                              .FORWARD_ONLY); // We have to
                                              // get all
                                              // records
            int count = 0;
            int price = 1;
            while (scrollableResults.next()) {
                Product product
                    = (Product)scrollableResults.get(0);
                product.setPrice(
                    price
                    + (price
                       * 10)); // update the price, this is
                               // just a sample
                price += 1;
                if (++count % batchSize
                    == 0) { // This is much required
                    session.flush();
                    session.clear();
                }
            }
            transaction.commit();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        finally {
            if (scrollableResults != null) {
                scrollableResults.close();
            }
            if (session != null) {
                session.close();
            }
        }
        HibernateUtil.shutdown();
    }
}
```



```
[main] DEBUG org.hibernate.engine.jdbc.batch.internal.BatchingBatch - Executing batch size: 10
[main] DEBUG org.hibernate.engine.jdbc.batch.internal.BatchingBatch - Executing batch size: 10
[main] DEBUG org.hibernate.engine.jdbc.batch.internal.BatchingBatch - Executing batch size: 10
[main] DEBUG org.hibernate.engine.jdbc.batch.internal.BatchingBatch - Executing batch size: 10
[main] DEBUG org.hibernate.engine.jdbc.batch.internal.BatchingBatch - Executing batch size: 10
[main] DEBUG org.hibernate.engine.jdbc.batch.internal.BatchingBatch - Executing batch size: 10
[main] DEBUG org.hibernate.engine.jdbc.batch.internal.BatchingBatch - Executing batch size: 10
[main] DEBUG org.hibernate.engine.jdbc.batch.internal.BatchingBatch - Executing batch size: 10
[main] DEBUG org.hibernate.engine.jdbc.batch.internal.BatchingBatch - Executing batch size: 10
[main] DEBUG org.hibernate.engine.jdbc.batch.internal.BatchingBatch - Executing batch size: 10
```


MySQL output is as follows:



id	price	productBrand	productName
1	11	A	Product1
2	22	A	Product2
3	33	A	Product3
4	44	A	Product4
5	55	A	Product5
6	66	A	Product6
7	77	A	Product7
8	88	A	Product8
9	99	A	Product9
10	110	A	Product10
11	121	A	Product11

Conclusion

By using "**hibernate.jdbc.batch_size**" we can enable batch processing in hibernate session.flush() and session.clear() should be periodically done

to avoid exceptions and it is a good practice essentially required for batch processing.