Quarkus - Using Blaze-Persistence

Blaze-Persistence offers a fluent query builder API on top of JPA with a deep Hibernate ORM integration that enables the use of advanced SQL features like Common Table Expressions while staying in the realm of the JPA model.

On top of that, the Blaze-Persistence Entity-View module allows for DTO definitions that can be applied to business logic queries which are then transformed to optimized queries that only fetch the data that is needed to construct the DTO instances. The same DTO definitions can further be used for applying database updates, leading to a great reduction in boilerplate code and removing the need for object mapping tools.

> ℹ This extension is developed by a third party and is part of the Quarkus Platform.

Setting up and configuring Blaze-Persistence

The extension comes with default producers for `CriteriaBuilderFactory` and `EntityViewManager` that work out of the box given a working Hibernate ORM configuration. For customization, overriding of the default producers is possible via the standard mechanism as documented in the Quarkus CDI reference. This is needed if you need to set custom Blaze-Persistence properties.

In Quarkus, you just need to:

- `@Inject CriteriaBuilderFactory` or `EntityViewManager` and use it

- annotate your entity views with `@EntityView` and any other mapping annotation as usual

Add the following dependencies to your project:

- the Blaze-Persistence extension: `com.blazebit:blaze-persistence-integration-quarkus`

- further Blaze-Persistence integrations as needed:
  - `blaze-persistence-integration-jackson` for Jackson
  - `blaze-persistence-integration-jaxrs` for JAX-RS

Example dependencies using Maven

```
<dependencies>
    <!-- Blaze-Persistence specific dependencies -->
    <dependency>
        <groupId>com.blazebit</groupId>
        <artifactId>blaze-persistence-integration-
quarkus</artifactId>
    </dependency>
    <dependency>
        <groupId>com.blazebit</groupId>
        <artifactId>blaze-persistence-integration-hibernate-
5.4</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

The use in native images requires a dependency on the entity view annotation processor that may be extracted into a separate `native` profile:

```
<profiles>
    <profile>
        <id>native</id>
        <dependencies>
            <dependency>
                <groupId>com.blazebit</groupId>
                <artifactId>blaze-persistence-entity-view-
processor</artifactId>
                <scope>provided</scope>
            </dependency>
        </dependencies>
    </profile>
</profiles>
```

A `CriteriaBuilderFactory` and an `EntityViewManager` will be created based on the configured `EntityManagerFactory` as provided by the Hibernate-ORM extension.

You can then access these beans via injection:

Example application bean using Hibernate

```
@ApplicationScoped
public class SantaClausService {
    @Inject
    EntityManager em;  ①
    @Inject
    CriteriaBuilderFactory cbf;  ②
    @Inject
    EntityViewManager evm;  ③

    @Transactional  ④
    public List<GiftView> findAllGifts() {
        CriteriaBuilder<Gift> cb = cbf.create(em, Gift.class);
        return
emv.applySetting(EntityViewSetting.create(GiftView.class),
cb).getResultList();
    }
}
```

① Inject the `EntityManager`

② Inject the `CriteriaBuilderFactory`

③ Inject the `EntityViewManager`

④ Mark your CDI bean method as `@Transactional` so that a transaction is started or joined.

Example Entity

```java
@Entity
public class Gift {
    private Long id;
    private String name;
    private String description;

    @Id @GeneratedValue(strategy = GenerationType.SEQUENCE,
generator="giftSeq")
    public Long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }


    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
}
```

Example Entity-View

```java
@EntityView(Gift.class)
public interface GiftView {

    @IdMapping
    Long getId();

    String getName();
}
```

Example updatable Entity-View

```
@UpdatableEntityView
@CreatableEntityView
@EntityView(Gift.class)
public interface GiftUpdateView extends GiftView {

    void setName(String name);
}
```

Example JAX-RS Resource

```java
@Path("/gifts")
public class GiftResource {
    @Inject
    EntityManager entityManager;
    @Inject
    EntityViewManager entityViewManager;
    @Inject
    SantaClausService santaClausService;

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    @Transactional
    public Response createGift(GiftUpdateView view) {
        entityViewManager.save(entityManager, view);
        return Response.created(URI.create("/gifts/" +
view.getId())).build();
    }

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<GiftView> getGifts() {
        return santaClausService.findAllGifts();
    }

    @PUT
    @Path("{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    @Transactional
    public GiftView updateGift(@EntityViewId("id") GiftUpdateView
view) {
        evm.save(em, view);
        return evm.find(em, GiftView.class, view.getId());
    }

    @Path("{id"})
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public GiftView getGift(@PathParam("id") Long id) {
        return return entityViewManager.find(entityManager,
GiftView.class, view.getId());
    }
}
```

Blaze-Persistence configuration properties

There are various optional properties useful to refine your `EntityViewManager` and `CriteriaBuilderFactory` or guide guesses of Quarkus.

There are no required properties, as long as the Hibernate ORM extension is configured properly.

When no property is set, the Blaze-Persistence defaults apply.

The configuration properties listed here allow you to override such defaults, and customize and tune various aspects.

🔒 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
🔒 `quarkus.blaze-persistence.template-eager-loading` A boolean flag to make it possible to prepare all view template caches on startup. By default the eager loading of the view templates is disabled to have a better startup performance. Valid values for this property are `true` or `false`.	boolean	
🔒 `quarkus.blaze-persistence.default-batch-size` An integer value that defines the default batch size for entity view attributes. By default the value is 1 and can be overridden either via `com.blazebit.persistence.view.BatchFetch#size()` or by setting this property via `com.blazebit.persistence.view.EntityViewSetting#setProperty`.	int	
🔒 `quarkus.blaze-persistence.expect-batch-mode` A mode specifying if correlation value, view root or embedded view batching is expected. By default the value is `values` and can be overridden by setting this property via `com.blazebit.persistence.view.EntityViewSetting#setProperty`. Valid values are - `values` - `view_roots` - `embedding_views`	string	
🔒 `quarkus.blaze-persistence.updater.eager-loading` A boolean flag to make it possible to prepare the entity view updater cache on startup. By default the eager loading of entity view updates is disabled to have a better startup performance. Valid values for this property are `true` or `false`.	boolean	

🔒 `quarkus.blaze-persistence.updater.disallow-owned-updatable-subview` A boolean flag to make it possible to disable the strict validation that disallows the use of an updatable entity view type for owned relationships. By default the use is disallowed i.e. the default value is `true`, but since there might be strange models out there, it possible to allow this. Valid values for this property are `true` or `false`.	boolean	
🔒 `quarkus.blaze-persistence.updater.strict-cascading-check` A boolean flag to make it possible to disable the strict cascading check that disallows setting updatable or creatable entity views on non-cascading attributes before being associated with a cascading attribute. When disabled, it is possible, like in JPA, that the changes done to an updatable entity view are not flushed when it is not associated with an attribute that cascades updates. By default the use is enabled i.e. the default value is `true`. Valid values for this property are `true` or `false`.	boolean	
🔒 `quarkus.blaze-persistence.updater.error-on-invalid-plural-setter` A boolean flag that allows to switch from warnings to boot time validation errors when invalid plural attribute setters are encountered while the strict cascading check is enabled. When `true`, a boot time validation error is thrown when encountering an invalid setter, otherwise just a warning. This configuration has no effect when the strict cascading check is disabled. By default the use is disabled i.e. the default value is `false`. Valid values for this property are `true` or `false`.	boolean	
🔒 `quarkus.blaze-persistence.create-empty-flat-views` A boolean flag that allows to specify if empty flat views should be created by default if not specified via `EmptyFlatViewCreation`. By default the creation of empty flat views is enabled i.e. the default value is `true`. Valid values for this property are `true` or `false`.	boolean	
🔒 `quarkus.blaze-persistence.expression-cache-class` The full qualified expression cache implementation class name.	string	

🔒 `quarkus.blaze-persistence.inline-ctes` If set to true, the CTE queries are inlined by default. Valid values for this property are `true`, `false` or `auto`. Default is `true` which will always inline non-recursive CTEs. The `auto` configuration will only make use of inlining if the JPA provider and DBMS dialect support/require it. The property can be changed for a criteria builder before constructing a query.	boolean	

Limitations

Apache Derby

Blaze-Persistence currently does not come with support for Apache Derby. This limitation could be lifted in the future, if there's a compelling need for it and if someone contributes it.