### When to Use Custom Repositories

**1. Advanced Queries**

While simple queries can be written directly in JpaRepository interfaces, complex queries often require more sophisticated techniques. Custom repositories are ideal for scenarios where:

* **Complex Query Logic:** When JPQL or SQL queries involve complex joins, subqueries, or dynamic query building.
* **Advanced Features:** If you need advanced features like pagination, sorting, or criteria-based queries not easily supported by method naming conventions.

**2. Dynamic Queries**

When queries need to be built dynamically based on varying conditions, custom repositories are useful.

**3. Performance Optimization**

In performance-critical applications, custom repositories allow you to fine-tune queries and use database features that JpaRepository might not expose.

### Why Use Custom Repositories?

Here’s a detailed comparison of standard JPA repository methods versus custom repositories and their use cases:

#### 1. **Complex Queries**

**Standard JPA Repository Methods:**

* **Simple Queries:** Methods like findByStatusAndAmountGreaterThan work well for straightforward queries.
* **Method Naming Conventions:** Limited to queries that can be expressed through method names.

**Custom Repository Methods:**

* **Complex Queries:** Use JPQL or native SQL for complex queries with joins, subqueries, or aggregations.

**Example JPA :**

**public List<Order> findOrdersWithJoinsAndAggregations();**

**Example Custom repo :**

**String jpql = "SELECT o FROM Order o JOIN o.customer c WHERE o.status = :status AND o.amount > :amount";**

#### 2. **Dynamic Queries**

**Standard JPA Repository Methods:**

* **Static Queries:** Query conditions are fixed.

**Custom Repository Methods:**

* **Dynamic Queries:** Construct queries based on runtime conditions.

#### 3. **Performance Optimization**

**Standard JPA Repository Methods:**

* **Basic Queries:** Limited to standard CRUD and simple queries.

**Custom Repository Methods:**

* **Performance Tuning:** Optimize queries with advanced features.

#### 4. **Reusability and Separation of Concerns**

**Standard JPA Repository Methods:**

* **Limited Reusability:** Methods are bound to the repository interface.

**Custom Repository Methods:**

* **Enhanced Reusability:** Encapsulate complex logic and reuse it across different parts of the application.

### Custom Repository Examples in Practice

Let’s see a deeper example where using a custom repository offers significant benefits:

#### Example 1: **Dynamic Filtering with Criteria API**

**Scenario:** Fetch orders with various optional filters.

**Custom Repository Method:**

public List<Order> findOrdersWithDynamicFilters(OrderFilter filter);

Implementation:

public List<Order> findOrdersWithDynamicFilters(OrderFilter filter) {

CriteriaBuilder cb = entityManager.getCriteriaBuilder();

CriteriaQuery<Order> query = cb.createQuery(Order.class);

Root<Order> order = query.from(Order.class);

List<Predicate> predicates = new ArrayList<>();

if (filter.getStatus() != null) {

predicates.add(cb.equal(order.get("status"), filter.getStatus()));

}

if (filter.getAmount() != null) {

predicates.add(cb.greaterThan(order.get("amount"), filter.getAmount()));

}

query.where(predicates.toArray(new Predicate[0]));

return entityManager.createQuery(query).getResultList();

}

#### Example 2: **Complex Aggregation Query**

**Scenario:** Get the total amount of orders by status.

**Custom Repository Method:**

public Map<String, BigDecimal> getTotalAmountByStatus();

Implementation:

public Map<String, BigDecimal> getTotalAmountByStatus() {

String jpql = "SELECT o.status, SUM(o.amount) FROM Order o GROUP BY o.status";

List<Object[]> results = entityManager.createQuery(jpql).getResultList();

Map<String, BigDecimal> resultMap = new HashMap<>();

for (Object[] result : results) {

resultMap.put((String) result[0], (BigDecimal) result[1]);

}

return resultMap;

}

**Comparing Standard and Custom Repository Approaches**

| **Aspect** | **Standard Repository Methods** | **Custom Repository Methods** |
| --- | --- | --- |
| **Complex Queries** | Limited to method naming conventions | Supports JPQL and SQL for complex queries |
| **Dynamic Queries** | Fixed conditions; limited flexibility | Dynamic queries based on conditions |
| **Performance Optimization** | Basic queries; relies on default strategies | Fine-tune queries for performance and specific needs |
| **Reusability** | Methods bound to the repository interface | Encapsulate complex logic; reusable across applications |
| **Advanced Features** | Basic querying features | Advanced features like criteria API, queryDSL |

### Conclusion

While JPA's repository methods cover many use cases, custom repositories become necessary for advanced features, dynamic queries, performance tuning, and complex logic. They provide flexibility and power to address scenarios beyond the capabilities of standard JpaRepository methods.