Criteria Query

```
CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();
CriteriaQuery<Task> criteriaQuery = criteriaBuilder.createQuery(Task.class);
Root<Task> root = criteriaQuery.from(Task.class);
```

Root class is class on which table we appy the criteria query

CriteriaBuilder class is present in the javax.presistence.*;

Start of the method

```
List<Predicate> predicateList = new ArrayList<>();
```

Write Logic Query in Between

```
Predicate[] predicateArr = new Predicate[predicateList.size()];
Predicate predicate = criteriaBuilder.and(predicateList.toArray(predicateArr));

criteriaQuery.where(predicate);
```

End of Method

```
taskHelper.prepareTaskCriteriaQuery(taskSearchCriteria, criteriaBuilder, criteriaQue
//set order
taskHelper.setOrder(taskSearchCriteria, root, criteriaQuery, criteriaBuilder);
TypedQuery<Task> typedQuery = entityManager.createQuery(criteriaQuery);

typedQuery.setFirstResult((int) pageable.getOffset());
typedQuery.setMaxResults(pageable.getPageSize());
long count = taskHelper.criteriaForCount(taskSearchCriteria);
Page<Task> taskPage = new PageImpl<>(typedQuery.getResultList(), pageable,count);
```

Run the query

Get Result

```
if (CoreUtil.isNonEmpty(taskSearchCriteria.getRoleId())) {
    predicateList.add(root.join("assignee").in(getUsersList(taskSearchCriteria.getRoleId())));
}
```

In this we get the List<Integer> of the RoleId so we need to apply the filter on the mulitselect dropdown o Role

method-> in()

```
if (CoreUtil.isNonEmpty(taskSearchCriteria.getStatusList())) {
    predicateList
            .add(root.get("workflow").get("status").get("statusId").in(taskSearchCriteria.getStatusList()));
}

if (CoreUtil.isNonEmpty(taskSearchCriteria.getAccountComplexityIdList())) {
    predicateList.add(
            root.get("workflow").get("accountComplexity").in(taskSearchCriteria.getAccountComplexityIdList()));
}
```

```
if (CoreUtil.isNonEmpty(taskSearchCriteria.getEmailRequest())
        && CoreUtil.isNonEmpty(taskSearchCriteria.getEmailRequest().getSubjectEmail())) {
    predicateList.add(criteriaBuilder.like(root.get("workflow").get("emailRequest").<String>get("subjectEmail"),
            "%" + taskSearchCriteria.getEmailRequest().getSubjectEmail() + "%"));
}
```

AS we need to search the data on the basis of Email Subject Line ....By using Like()

Similary find the employee who name start with 'A;
criteriaBuilder.like(root.get('employee').get('empName'),"A%"));

```
if (CoreUtil.isNonEmpty(taskSearchCriteria.getDueDate())) {
    LocalDate localDate = CoreUtil.convertUtcToIstZone(taskSearchCriteria.getDueDate());
    LocalDateTime beginning = localDate.minusDays(1).atTime(LocalTime.MAX);
    LocalDateTime end = localDate.atTime(23, 59, 59);
    predicateList.add(criteriaBuilder.between(root.get("workflow").get("dueDate"), beginning, end));
}
```

```
List<Predicate> predicateList = new ArrayList<>();
```

getDueDate()-> begDate,EndDate
By USing the Between()

```
if (CoreUtil.isNonEmpty(taskSearchCriteria.getWorkType()) && WorkTypeEnum.Additional_Rework.getId() =
    logger.info("taskSearchCriteria.getRequestType(): " + taskSearchCriteria.getWorkType());
    predicateList.add(criteriaBuilder.isTrue(root.get("workflow").get("additionalRework")));
}
```

```
    */
    if (!UIGridEnum.Archive_Task_Box.name().equals(taskSearchCriteria.getGridName())) {
        Predicate isActive = criteriaBuilder.equal(root.get("isActive"), true);
        Predicate isFinalTask = criteriaBuilder.equal(root.get("isFinalTask"), true);
        Predicate result = criteriaBuilder.or(isActive, isFinalTask);
        predicateList.add(result);
    }
```

Filter ->isActive ..(column (boolean)in Task table)

By using the Equal method()

```
private Predicate applyMyQcQueueScreenCrietria(TaskSearchCriteria taskSearchCriteria,
        CriteriaBuilder criteriaBuilder, Root<Task> root) {
    Predicate result = null;
    if (UIGridEnum.My_Qc_Task.name().equals(taskSearchCriteria.getGridName())) {
        Predicate taskName = criteriaBuilder.equal(root.get("taskName"), TaskActivityEnum.Review_submission.name());
        Predicate completedOn = criteriaBuilder.isNull(root.get("completedOn"));
        result = criteriaBuilder.and(taskName, completedOn);
    }
    return result;
}
```

Find the task of reviewer which is not completed as yet. equal() and isNull()
...creiteriaBuilder And() also.

```
predicateList.add(criteriaBuilder.greaterThanOrEqualTo(root.get("workflow").get("processorAllocationDate"),"2021-01-01"));
```

This is fetch the data on the basis of Alloaction Date whose has assgin task after 01 jan 2021

<N extends Number> Expression<N>	abs(Expression<N> x) Create an expression that returns the absolute value of its a
<Y> Expression<Y>	all(Subquery<Y> subquery) Create an all expression over the subquery results.
Predicate	and(Expression<Boolean> x, Expression<Boolean> Create a conjunction of the given boolean expressions.
Predicate	and(Predicate... restrictions) Create a conjunction of the given restriction predicates.
<Y> Expression<Y>	any(Subquery<Y> subquery) Create an any expression over the subquery results.
CompoundSelection<Object[]>	array(Selection<?>... selections) Create an array-valued selection item.
Order	asc(Expression<?> x) Create an ordering by the ascending value of the expressio
<N extends Number> Expression<Double>	avg(Expression<N> x) Create an aggregate expression applying the avg operation.
<Y extends Comparable<? super Y>> Predicate	between(Expression<? extends Y> v, Expression<? Create a predicate for testing whether the first argument is
<Y extends Comparable<? super Y>> Predicate	between(Expression<? extends Y> v, Y x, Y y) Create a predicate for testing whether the first argument is

Order	desc(Expression<?> x) Create an ordering by the descending value of the expression.
<N extends Number> Expression<N>	diff(Expression<? extends N> x, Expression<? exte Create an expression that returns the difference between its a
<N extends Number> Expression<N>	diff(Expression<? extends N> x, N y) Create an expression that returns the difference between its a
<N extends Number> Expression<N>	diff(N x, Expression<? extends N> y) Create an expression that returns the difference between its a
Predicate	disjunction() Create a disjunction (with zero disjuncts).
Predicate	equal(Expression<?> x, Expression<?> y) Create a predicate for testing the arguments for equality.
Predicate	equal(Expression<?> x, Object y) Create a predicate for testing the arguments for equality.
Predicate	exists(Subquery<?> subquery) Create a predicate testing the existence of a subquery result.

<Y extends Comparable<? super Y>> Predicate	greaterThan(Expression<? extends Y> x, Expression<? extends Y> y) Create a predicate for testing whether the first argument is greater than the second.
<Y extends Comparable<? super Y>> Predicate	greaterThan(Expression<? extends Y> x, Y y) Create a predicate for testing whether the first argument is greater than the second.
<Y extends Comparable<? super Y>> Predicate	greaterThanOrEqualTo(Expression<? extends Y> x, Expression<? extends Y> y) Create a predicate for testing whether the first argument is greater than or equal to the second.
<Y extends Comparable<? super Y>> Predicate	greaterThanOrEqualTo(Expression<? extends Y> x, Y y) Create a predicate for testing whether the first argument is greater than or equal to the second.
<X extends Comparable<? super X>> Expression<X>	greatest(Expression<X> x) Create an aggregate expression for finding the greatest of the values (strings, dates, etc).

<Y> Expression<Y>	nullif(Expression<Y> x, Y y) Create an expression that tests whether its argumen
<T> Expression<T>	nullLiteral(Class<T> resultClass) Create an expression for a null literal with the given
Predicate	or(Expression<Boolean> x, Expression<Bool Create a disjunction of the given boolean expressio
Predicate	or(Predicate... restrictions) Create a disjunction of the given restriction predica

<T> CriteriaBuilder.In<T>	in(Expression<? extends T> expression) Create predicate to test whether given expression is contain
<C extends Collection<?>> Predicate	isEmpty(Expression<C> collection) Create a predicate that tests whether a collection is empty.
Predicate	isFalse(Expression<Boolean> x) Create a predicate testing for a false value.
<E,C extends Collection<E>> Predicate	isMember(E elem, Expression<C> collection) Create a predicate that tests whether an element is a membe
<E,C extends Collection<E>> Predicate	isMember(Expression<E> elem, Expression<C> colle Create a predicate that tests whether an element is a membe
<C extends Collection<?>> Predicate	isNotEmpty(Expression<C> collection) Create a predicate that tests whether a collection is not emp
<E,C extends Collection<E>> Predicate	isNotMember(E elem, Expression<C> collection) Create a predicate that tests whether an element is not a me
<E,C extends Collection<E>> Predicate	isNotMember(Expression<E> elem, Expression<C> co Create a predicate that tests whether an element is not a me
Predicate	isNotNull(Expression<?> x) Create a predicate to test whether the expression is not null.
Predicate	isNull(Expression<?> x) Create a predicate to test whether the expression is null.
Predicate	isTrue(Expression<Boolean> x)

Predicate	like(Expression<String> x, Expression<String> pattern, Expressio Create a predicate for testing whether the expression satisfies the given pattern.
Predicate	like(Expression<String> x, String pattern) Create a predicate for testing whether the expression satisfies the given pattern.
Predicate	like(Expression<String> x, String pattern, char escapeChar) Create a predicate for testing whether the expression satisfies the given pattern.
Predicate	like(Expression<String> x, String pattern, Expression<Character> Create a predicate for testing whether the expression satisfies the given pattern.

<N extends Number> Expression<N>	max(Expression<N> x) Create an aggregate expression applying the numerical max operati
<N extends Number> Expression<N>	min(Expression<N> x) Create an aggregate expression applying the numerical min operati
Expression<Integer>	mod(Expression<Integer> x, Expression<Integer> y) Create an expression that returns the modulus of its arguments.
Expression<Integer>	mod(Expression<Integer> x, Integer y) Create an expression that returns the modulus of its arguments.

<Y extends Comparable<? super Y>> Predicate	lessThan(Expression<? extends Y> x, E Create a predicate for testing whether the firs
<Y extends Comparable<? super Y>> Predicate	lessThan(Expression<? extends Y> x, Y Create a predicate for testing whether the firs
<Y extends Comparable<? super Y>> Predicate	lessThanOrEqualTo(Expression<? exten Create a predicate for testing whether the firs
<Y extends Comparable<? super Y>> Predicate	lessThanOrEqualTo(Expression<? exten Create a predicate for testing whether the firs

Predicate	notEqual(Expression<?> x, Object y) Create a predicate for testing the arguments for inequality.
Predicate	notLike(Expression<String> x, Expression<String> pattern) Create a predicate for testing whether the expression does not satisfy the given pattern.
Predicate	notLike(Expression<String> x, Expression<String> pattern, char escapeCl Create a predicate for testing whether the expression does not satisfy the given pattern.
Predicate	notLike(Expression<String> x, Expression<String> pattern, Expression<Cl Create a predicate for testing whether the expression does not satisfy the given pattern.
Predicate	notLike(Expression<String> x, String pattern) Create a predicate for testing whether the expression does not satisfy the given pattern.