Dynamic SQL

It is general to reuse the defined query defined in the coding using JDBC API. However, there are cases that are difficult to solve only by changing the simple parameter values. In cases which require different query execution according to various conditions, there are problems which require connecting many if-else condition sections. Let's examine how to provide relatively flexible methods for the dynamic change of SQL sentence.

Using Basic Dynamic element

Refer to the use of sample Dynamic element below:

Sample Dynamic SQL mapping xml

```
..
<typeAlias alias="jobHistVO" type="egovframework.rte.psl.dataaccess.vo.JobHistVO" />

<select id="selectJobHistListUsingDynamicElement" parameterClass="jobHistVO"
resultClass="jobHistVO">
        <![CDATA[
                select EMP_NO       as empNo,
                       START_DATE as startDate,
                       END_DATE    as endDate,
                       JOB          as job,
                       SAL          as sal,
                       COMM         as comm,
                       DEPT_NO      as deptNo
                from    JOBHIST
        ]]>
        <dynamic prepend="where">
                <isNotNull property="empNo" prepend="and">
                        EMP_NO = #empNo#
                </isNotNull>
        </dynamic>
                order by EMP_NO, START_DATE
</select>
```

Above sql mapping file is the example of dynamically adding/removing where EMP_NO = #empNo# condition sentence depending on the existence of empNo property value of parameter objects. While "where" is designated as a prepend property of dynamic, if it does not satisfy one of lower element conditions, it is not added to sql sentence. In addition, above example designates prepend="and" to isNotNull tag as the lower elements, the prepend of condition that becomes the initial true is overwritten by 'where', the prepend of dynamic, and where EMP_NO = #empNo#.

Sample TestCase

```
..
    @Test
    public void testDynamicStatement() throws Exception {
        JobHistVO vo = new JobHistVO();
        // Dynamic test according to object property of the input parameter
        vo.setEmpNo(new BigDecimal(7788));

        // select
        List<JobHistVO> resultList =
            jobHistDAO.selectJobHistList(
                "selectJobHistListUsingDynamicElement", vo);

        // check
        assertNotNull(resultList);
        assertEquals(3, resultList.size());
```

```
        SimpleDateFormat sdf =
            new SimpleDateFormat("yyyy-MM-dd", java.util.Locale.getDefault());
        assertEquals(sdf.parse("1987-04-19"), resultList.get(0).getStartDate());
        assertEquals(sdf.parse("1988-04-13"), resultList.get(1).getStartDate());
        assertEquals(sdf.parse("1990-05-05"), resultList.get(2).getStartDate());

        // Dynamic test according to object property of the input parameter
        vo.setEmpNo(null);

        // select
        resultList =
            jobHistDAO.selectJobHistList(
                "selectJobHistListUsingDynamicElement", vo);

        // check
        assertNotNull(resultList);
        // Overall data will be searched since where in not executed
        assertEquals(17, resultList.size());

    }
```

When setting the value of empNo in parameter object as above, only 3 cases of Job history for employee number 7788 is inquired which the relevant condition is added actively. However, all histories for all employees is inquired when the empNo value is not set (same as setting as null).

Comparing Unary conditions

Refer to the example of sample sql mapping xml below.

In general, the dynamic element to write Dynamic SQL is used frequently to change “where”. However, below is an example that utilized Unary comparison calculation when re-inquiring results transmitted by arbitrary constant on the dual for test purposes.

Sample Unary Comparison

```
        ..
        <typeAlias alias="egovMap" type="egovframework.rte.psl.dataaccess.util.EgovMap" />

        <select id="selectDynamicUnary" parameterClass="map" remapResults="true"
resultClass="egovMap">
                select
                <dynamic>
                        <isEmpty property="testEmptyString">
                                'empty String' as IS_EMPTY_STRING
                        </isEmpty>
                        <isNotEmpty property="testEmptyString">
                                'not empty String' as IS_EMPTY_STRING
                        </isNotEmpty>
                        <isEmpty prepend=", " property="testEmptyCollection">
                                'empty Collection' as IS_EMPTY_COLLECTION
                        </isEmpty>
                        <isNotEmpty prepend=", " property="testEmptyCollection">
                                'not empty Collection' as IS_EMPTY_COLLECTION
                        </isNotEmpty>
                        <isNull prepend=", " property="testNull">
                                'null' as IS_NULL
                        </isNull>
                        <isNotNull prepend=", " property="testNull">
                                'not null' as IS_NULL
                        </isNotNull>
                        <isPropertyAvailable prepend=", " property="testProperty">
                                'testProperty Available' as  TEST_PROPERTY_AVAILABLE
```

```
                    </isPropertyAvailable>
                    <isNotPropertyAvailable prepend=", " property="testProperty">
                        'testProperty Not Available' as TEST_PROPERTY_AVAILABLE
                    </isNotPropertyAvailable>
                </dynamic>
                from dual
        </select>
```

The Unary comparison and calculation tag tested above is as following.

- isEmpty : In case the Collection, String(or String.valueOf()) properties is null or empty("" or size() < 1), it is true
- isNotEmpty : In case the Collection, String(or String.valueOf()) properties is null or empty("" or size() < 1), it is true
- isNull : In case the related properties is null, it is true
- isNotNull : In case the related properties is not null, it is true
-
- isPropertyAvailable : In case there is relevant properties are existent on the parameter object, it is true.
- isNotPropertyAvailable : In case there is no relevant properties on the parameter object, it is true.

Properties used in the Unary comparison calculation tag is as following.

- prepend : SQL area possible to override added in front of active syntax.
- property : Designate the properties to be used in essential parameter object.
- removeFirstPrepend : Whether to remove the prepend of the tag that creates the content included first (true/false)
- open : Open string for overall result syntax
- close : Close string on overall result syntax

Sample TestCase

```
..
    @SuppressWarnings("unchecked")
    @Test
    public void testDynamicUnary() throws Exception {
        Map map = new HashMap();
        // Dynamic test according to properties of input parameter object
        // isEmpty test - String
        map.put("testEmptyString", "");
        // isEmpty test - Collection
        List list = new ArrayList();
        map.put("testEmptyCollection", list);
        // isNull test
        map.put("testNull", null);
        // isPropertyAvailable test - cf.) Note that the relevant properties is available when the
property value is configured null.
        map.put("testProperty", null);

        // select
        Map resultMap =
            (Map) jobHistDAO.getSqlMapClientTemplate().queryForObject(
                "selectDynamicUnary", map);

        // check
        assertNotNull(resultMap);
        assertEquals("empty String", resultMap.get("isEmptyString"));
        assertEquals("empty Collection", resultMap.get("isEmptyCollection"));
        assertEquals("null", resultMap.get("isNull"));
        assertEquals("testProperty Available", resultMap
            .get("testPropertyAvailable"));
```

```
        // Dynamic Test 2 according to the property of the input parameter object
        // isEmpty Tes - isEmpty satisfies String-null
        map.put("testEmptyString", null);
        // isEmpty Test- Collection - isEmpty is satisfied in case of null
        List nullList = null;
        map.put("testEmptyCollection", nullList);

        // select
        resultMap =
            (Map) jobHistDAO.getSqlMapClientTemplate().queryForObject(
                "selectDynamicUnary", map);

        // check
        assertNotNull(resultMap);
        assertEquals("empty String", resultMap.get("isEmptyString"));
        assertEquals("empty Collection", resultMap.get("isEmptyCollection"));

        // Dynamic Test 2 according to the property of the input parameter object
        map.clear();
        // isEmpty test - String
        map.put("testEmptyString", "aa");
        // isEmpty test - Collection
        list.clear();
        list.add("aa");
        map.put("testEmptyCollection", list);
        // isNull test
        map.put("testNull", new BigDecimal(0));
        // isPropertyAvailable test - isNotPropertyAvailable when the key itself is not contained
        // map.put("testProperty", null);

        // select
        resultMap =
            (Map) jobHistDAO.getSqlMapClientTemplate().queryForObject(
                "selectDynamicUnary", map);

        // check
        assertNotNull(resultMap);
        assertEquals("not empty String", resultMap.get("isEmptyString"));
        assertEquals("not empty Collection", resultMap.get("isEmptyCollection"));
        assertEquals("not null", resultMap.get("isNull"));
        assertEquals("testProperty Not Available", resultMap
            .get("testPropertyAvailable"));

    }
```

This is the example of testing which condition satisfies which cases by setting the input parameter object (map, in this case) diversely to compare the unary conditions. For isEmpty, String is null “”, or the subsequent element on the collection is not added. When the collection object itself is null, it satisfies all. The isPropertyAvailable tag only adds the relevant key on the input object. The value is true when it is null. It is used most frequently in simple comparison of isNotNull or isNotEmpty on certain properties of factors in case the where active condition syntax of Dynamic SQL is changed.

Binary Condition Comparison

Refer to the example of sample sql mapping xml below.

Similarly, below is an example that utilized binary comparison calculation in the process of re-inquiring the result transmitted for test purposes.

Sample Binary Comparison

```
..
	<typeAlias alias="egovMap" type="egovframework.rte.psl.dataaccess.util.EgovMap" />

	<select id="selectDynamicBinary" parameterClass="map" remapResults="true"
resultClass="egovMap">
		select
		<dynamic>
			<isEqual property="testString" compareValue="test">
				'$testString$' as TEST_STRING, 'test : equals' as IS_EQUAL
			</isEqual>
			<isNotEqual property="testString" compareValue="test">
				'$testString$' as TEST_STRING, 'test : not equals' as IS_EQUAL
			</isNotEqual>
			<isPropertyAvailable property="testNumeric">
				<isEqual property="testNumeric" prepend=", "
compareValue="10">
					cast($testNumeric$ as $castTypeScale$) as
TEST_NUMERIC, '10 : equals' as IS_EQUAL_NUMERIC
				</isEqual>
				<isNotEqual property="testNumeric" prepend=", "
compareValue="10">
					cast($testNumeric$ as $castTypeScale$) as
TEST_NUMERIC, '10 : not equals' as IS_EQUAL_NUMERIC
				</isNotEqual>
			</isPropertyAvailable>
			<isGreaterEqual property="testNumeric" prepend=", "
compareValue="10">
				'10 <![CDATA[<=]]> $testNumeric$' as IS_GREATER_EQUAL
			</isGreaterEqual>
			<isGreaterThan property="testNumeric" prepend=", "
compareValue="10">
				'10 <![CDATA[<]]> $testNumeric$' as IS_GREATER_THAN
			</isGreaterThan>
			<isLessEqual property="testNumeric" prepend=", " compareValue="10">
				'10 <![CDATA[>=]]> $testNumeric$' as IS_LESS_EQUAL
			</isLessEqual>
			<isLessThan property="testNumeric" prepend=", " compareValue="10">
				'10 <![CDATA[>]]> $testNumeric$' as IS_LESS_THAN
			</isLessThan>
			<!-- checkMore -->
			<isPropertyAvailable property="testOtherString">
				<isEqual property="testOtherString" prepend=", "
compareProperty="testString">
					'$testOtherString$' as TEST_OTHER_STRING, 'test :
testOtherString equals testString' as COMPARE_PROPERTY_EQUAL
				</isEqual>
				<isNotEqual property="testOtherString" prepend=", "
compareProperty="testString">
					'$testOtherString$' as TEST_OTHER_STRING, 'test :
testOtherString not equals testString' as COMPARE_PROPERTY_EQUAL
				</isNotEqual>
				<isGreaterEqual property="testOtherString" prepend=", "
compareProperty="testString">
					'''$testOtherString$'' <![CDATA[>=]]> ''$testString$''' as
COMPARE_PROPERTY_GREATER_EQUAL
				</isGreaterEqual>
				<isGreaterThan property="testOtherString" prepend=", "
compareProperty="testString">
					'''$testOtherString$'' <![CDATA[>]]> ''$testString$''' as
COMPARE_PROPERTY_GREATER_THAN
				</isGreaterThan>
				<isLessEqual property="testOtherString" prepend=", "
compareProperty="testString">
```

```
                                        '''$testOtherString$'' <![CDATA[<=]]> ''$testString$''' as
COMPARE_PROPERTY_LESS_EQUAL
                                    </isLessEqual>
                                    <isLessThan property="testOtherString" prepend=", "
compareProperty="testString">
                                        '''$testOtherString$'' <![CDATA[<]]> ''$testString$''' as
COMPARE_PROPERTY_LESS_THAN
                                    </isLessThan>
                                </isPropertyAvailable>
                            </dynamic>
                            from dual
                    </select>
```

The tested binary comparison calculation tag is as below.

- isEqual : true when the property is equal to the compareValue or compareProperty
- isNotEqual : true when the property is different from the compareValue or compareProperty
- isGreaterEqual : true when the property is same or greater than the compareValue or compareProperty
- isGreaterThan : true when the property is greater as the compareValue or compareProperty
- isLessEqual : true when the property equals to or is less than the compareValue or compareProperty
- isLessThan : true when the property is less than the compareValue or compareProperty

The properties that can be used in binary comparison calculation tag is as following.

- prepend : SQL area which enables the override added in front of active syntax
- property : Mandatory. Designate the properties to which the parameter object is compared.
- compareProperty : Designate to compare values between other properties and relevant properties of the parameter object (mandatory when there is no compareValue)
- compareValue : Desginate the value to be compared to the relevant property (mandatory when there is no compareValue)

- removeFirstPrepend : Whether to remove the prepend of the tag that creates the content included first (true/false)
- open : Open string for overall result syntax

- close : Close string on overall result syntax

It can be confirmed that each comparison calculation tag above is repeated. When it requires complex processing of conditions, it could be configured appropriately by tagging various comparison calculation. Moreover, when it requires the escape processing on XML such as comparison operators (>, <), it can be bound as CDATA sections as <![CDATA[>]]>. Although it will be easier to bind the CDATA section on the overall query sections, the dynamic element itself should be interpreted as the actual XML tag. Therefore, note that there is difficulty in using the special characters individually occurred within the dynamic area. cf.) Instead of < , >, the escape can be done directly such as < and >.

Sample TestCase

```
..
    @SuppressWarnings("unchecked")
    @Test
    public void testDynamicBinary() throws Exception {
        Map map = new HashMap();
        String castTypeScale = "numeric(2)";
        // In oracle, - type corresponding to numeric is number
        if (isOracle) {
            castTypeScale = "number(2)";
        } else if (isMysql) {
            castTypeScale = "decimal(2)";
        }
```

```
        // Dynamic test according to properties of input parameter objects.
        // isEqual test - String
        map.put("testString", "test");
        // isEqual test - BigDecimal
        map.put("testNumeric", new BigDecimal(10));
        // cast processing is added for numeric(db) - decimal(java) processing in inquiring constant on
the temporary dual table.
        map.put("castTypeScale", castTypeScale);

        // select
        Map resultMap =
            (Map) jobHistDAO.getSqlMapClientTemplate().queryForObject(
                "selectDynamicBinary", map);

        // check
        assertNotNull(resultMap);
        assertEquals("test", resultMap.get("testString"));
        assertEquals("test : equals", resultMap.get("isEqual"));
        assertEquals(new BigDecimal(10), resultMap.get("testNumeric"));
        assertEquals("10 : equals", resultMap.get("isEqualNumeric"));
        assertEquals("10 <= 10", resultMap.get("isGreaterEqual"));
        assertTrue(!resultMap.containsKey("isGreaterThan"));
        assertEquals("10 >= 10", resultMap.get("isLessEqual"));
        assertTrue(!resultMap.containsKey("isLessThan"));

        // Dynamic test 2 according to properties of input parameter objects.
        map.clear();

        // isEqual test - String
        map.put("testString", "not test");
        // isEqual test - BigDecimal
        map.put("testNumeric", new BigDecimal(11));
        // cast processing is added for numeric(db) - decimal(java) processing in inquiring constant on
the temporary dual table.
        map.put("castTypeScale", castTypeScale);

        // select
        resultMap =
            (Map) jobHistDAO.getSqlMapClientTemplate().queryForObject(
                "selectDynamicBinary", map);

        // check
        assertNotNull(resultMap);
        assertEquals("not test", resultMap.get("testString"));
        assertEquals("test : not equals", resultMap.get("isEqual"));
        assertEquals(new BigDecimal(11), resultMap.get("testNumeric"));
        assertEquals("10 : not equals", resultMap.get("isEqualNumeric"));
        assertEquals("10 <= 11", resultMap.get("isGreaterEqual"));
        assertEquals("10 < 11", resultMap.get("isGreaterThan"));
        assertTrue(!resultMap.containsKey("isLessEqual"));
        assertTrue(!resultMap.containsKey("isLessThan"));

        // Dynamic test 2 according to properties of input parameter objects
        map.clear();

        // isEqual test - String
        // It is matched with isNotEqual without generating errors when the null value is transmitted to
the property which is the subject of isEqual comparison.
        map.put("testString", null);
        // isEqual test - BigDecimal
        map.put("testNumeric", new BigDecimal(9));
```

```
        // cast processing is added for numeric(db) - decimal(java) processing in inquiring constant on
the temporary dual table.
        map.put("castTypeScale", castTypeScale);

        // select
        resultMap =
            (Map) jobHistDAO.getSqlMapClientTemplate().queryForObject(
                "selectDynamicBinary", map);

        // check
        assertNotNull(resultMap);
        // In oracle, “ equals to null and the result object is mapped as null.
        assertEquals(!(isOracle || isTibero) ? "" : null, resultMap
            .get("testString"));
        assertEquals("test : not equals", resultMap.get("isEqual"));
        assertEquals(new BigDecimal(9), resultMap.get("testNumeric"));
        assertEquals("10 : not equals", resultMap.get("isEqualNumeric"));
        assertTrue(!resultMap.containsKey("isGreaterEqual"));
        assertTrue(!resultMap.containsKey("isGreaterThan"));
        assertEquals("10 >= 9", resultMap.get("isLessEqual"));
        assertEquals("10 > 9", resultMap.get("isLessThan"));

        // Dynamic test 3 according to properties of input parameter objects
        map.clear();

        map.put("testString", "test");
        // isEqual test - BigDecimal
        map.put("testOtherString", "test");

        // select
        resultMap =
            (Map) jobHistDAO.getSqlMapClientTemplate().queryForObject(
                "selectDynamicBinary", map);

        // check
        assertNotNull(resultMap);

        assertEquals("test : equals", resultMap.get("isEqual"));
        // Expected result when the testNumeric property is not
        assertTrue(!resultMap.containsKey("isGreaterEqual"));
        assertTrue(!resultMap.containsKey("isGreaterThan"));
        assertTrue(!resultMap.containsKey("isLessEqual"));
        assertTrue(!resultMap.containsKey("isLessThan"));
        // compare testOtherString
        assertEquals("test", resultMap.get("testOtherString"));
        assertEquals("test : testOtherString equals testString", resultMap
            .get("comparePropertyEqual"));
        assertEquals("'test' >= 'test'", resultMap
            .get("comparePropertyGreaterEqual"));
        assertTrue(!resultMap.containsKey("comparePropertyGreaterThan"));
        assertEquals("'test' <= 'test'", resultMap
            .get("comparePropertyLessEqual"));
        assertTrue(!resultMap.containsKey("comparePropertyLessThan"));

        // Dynamic test 4 according to properties of input parameter objects
        map.clear();

        map.put("testString", "test");
        // 'test' >= 'sample' test
        map.put("testOtherString", "sample");

        // select
        resultMap =
```

```
            (Map) jobHistDAO.getSqlMapClientTemplate().queryForObject(
                "selectDynamicBinary", map);

        // check
        assertNotNull(resultMap);

        assertEquals("test : equals", resultMap.get("isEqual"));
        // expected result not exceeding the testNumeric property
        assertTrue(!resultMap.containsKey("isGreaterEqual"));
        assertTrue(!resultMap.containsKey("isGreaterThan"));
        assertTrue(!resultMap.containsKey("isLessEqual"));
        assertTrue(!resultMap.containsKey("isLessThan"));
        // testOtherString compare
        assertEquals("sample", resultMap.get("testOtherString"));
        assertEquals("test : testOtherString not equals testString", resultMap
            .get("comparePropertyEqual"));
        assertTrue(!resultMap.containsKey("comparePropertyGreaterEqual"));
        assertTrue(!resultMap.containsKey("comparePropertyGreaterThan"));
        assertEquals("'sample' <= 'test'", resultMap
            .get("comparePropertyLessEqual"));
        assertEquals("'sample' < 'test'", resultMap
            .get("comparePropertyLessThan"));

        // Dynamic test 5 according to properties of input parameter objects
        map.clear();

        map.put("testString", "test");
        // 'test' <= 'testa' test
        map.put("testOtherString", "testa");

        // select
        resultMap =
            (Map) jobHistDAO.getSqlMapClientTemplate().queryForObject(
                "selectDynamicBinary", map);

        // check
        assertNotNull(resultMap);

        assertEquals("test : equals", resultMap.get("isEqual"));
        // expected result not exceeding the testNumeric property
        assertTrue(!resultMap.containsKey("isGreaterEqual"));
        assertTrue(!resultMap.containsKey("isGreaterThan"));
        assertTrue(!resultMap.containsKey("isLessEqual"));
        assertTrue(!resultMap.containsKey("isLessThan"));
        // compare testOtherString
        assertEquals("testa", resultMap.get("testOtherString"));
        assertEquals("test : testOtherString not equals testString", resultMap
            .get("comparePropertyEqual"));
        assertEquals("'testa' >= 'test'", resultMap
            .get("comparePropertyGreaterEqual"));
        assertEquals("'testa' > 'test'", resultMap
            .get("comparePropertyGreaterThan"));
        assertTrue(!resultMap.containsKey("comparePropertyLessEqual"));
        assertTrue(!resultMap.containsKey("comparePropertyLessThan"));

    }
```

This is the example of testing which condition satisfies which cases by setting the input parameter object (map, in this case) diversely to compare the binary conditions. In case of isGreaterEqual, isGreaterThan, isLessEqual, isLessThan comparison on the input object properties of the number type, the result can be anticipated easily, and for number format result inquiry, the cast on the DB column is made to be processed. It is confirmed that isGreaterEqual, isGreaterThan, isLessEqual, isLessThan is possible on the String in Scenario 4 and 5, and it is 'sample' < 'test' , 'testa' > 'test'.

ParameterPresent Comparison

Refer to example of sample sql mapping xml below.

Sample ParameterPresent Comparison

```
..
<typeAlias alias="egovMap" type="egovframework.rte.psl.dataaccess.util.EgovMap" />

<select id="selectDynamicParameterPresent" parameterClass="map" remapResults="true"
resultClass="egovMap">
        select
                <isParameterPresent>
                        'parameter object exist' as IS_PARAMETER_PRESENT
                </isParameterPresent>
                <isNotParameterPresent>
                        'parameter object not exist' as IS_PARAMETER_PRESENT
                </isNotParameterPresent>
        from dual
</select>
```

- isParameterPresent : true when the parameter object is transmitted (not null)
- isNotParameterPresent : true when the parameter object is not transmitted (null)

Properties used on the ParameterPresent comparison calculation tag is as following.

- prepend : SQL area which enables the override added in front of active syntax
- removeFirstPrepend : Whether to remove the prepend of the tag that creates the content included first (true/false)
- open : Open string for overall result syntax
- close : Close string on overall result syntax

Sample TestCase

```
..
    @SuppressWarnings("unchecked")
    @Test
    public void testDynamicParameterPresent() throws Exception {

        // Dynamic test according to properties of input parameter objects
        // isParameterPresent test
        Map map = new HashMap();

        // select
        Map resultMap =
            (Map) jobHistDAO.getSqlMapClientTemplate().queryForObject(
                "selectDynamicParameterPresent", map);

        // check
        assertNotNull(resultMap);
        assertEquals("parameter object exist", resultMap
            .get("isParameterPresent"));

        map = null;

        // select
        resultMap =
```

```
            (Map) jobHistDAO.getSqlMapClientTemplate().queryForObject(
                "selectDynamicParameterPresent", map);

        // check
        assertNotNull(resultMap);
        assertEquals("parameter object not exist", resultMap
            .get("isParameterPresent"));
    }
```

In calling API for executing queries of iBATIS, the comparison calculation of isParameterPresent, isNotParameterPresent can be used according to the transmission of parameter objects.

iterate Calculation

Refer to example of sample sql mapping xml below.

This is most frequently used example of in condition syntax processing for iterate tag processing.

Sample iterate Calculation

```
        ..
        <typeAlias alias="jobHistVO" type="egovframework.rte.psl.dataaccess.vo.JobHistVO" />
        <typeAlias alias="empIncludesEmpListVO"
type="egovframework.rte.psl.dataaccess.vo.EmpIncludesEmpListVO" />

        <select id="selectJobHistListUsingDynamicIterate" parameterClass="empIncludesEmpListVO"
resultClass="jobHistVO">
                <![CDATA[
                        select EMP_NO       as empNo,
                                START_DATE as startDate,
                                END_DATE    as endDate,
                                JOB          as job,
                                SAL          as sal,
                                COMM         as comm,
                                DEPT_NO     as deptNo
                        from    JOBHIST
                ]]>
                <dynamic prepend="where">
                        <iterate property="empList" open="EMP_NO in (" conjunction=", "
close=")">
                                #empList[].empNo#
                        </iterate>
                </dynamic>
                        order by EMP_NO, START_DATE
        </select>
```

- iterate : Execute related contents by rotating the repeated loop for each individual element included in the collection format object

Following is the properties that can be used at iterate tag.

- prepend : SQL area which enables the override added in front of active syntax
- property : When the object of java.util.Collection, java.util.Iterator, or array type is not designated or indicated, it assumes that the parameter object as a collection.

- removeFirstPrepend : Whether to remove the prepend of the tag that creates the content included first (true/false/iterate)
- open : Open string for overall result syntax

- close : Close string on overall result syntax

- conjunction : Useful when the string applied in between each iteration, operator such as 'and' and 'or', and separators such as ','.

EmpIncludesEmpListVO that includes the list object which the List<EmpVO> is used as the parameter object. The empNo value of the EmpVO object is created actively by the iterate tag.

where EMP_NO in (? , ? , ?)

Now the body area indication on the iterate tag. As confirmed in #empList[].empNo#, the 'list property name []' is used to indicate current item within the looping. Here, the relevant item is EmpVO and it contains the empNo property. And this value is bound as current item in the in phase. If the parameter object itself is possible to iterate, the current item can be designated on the body area without indicating iterate property.

- In the iBATIS, dot notation ('.') is used when the parameter object is a compound object to approach sub object or properties.

Sample TestCase

..

```
@SuppressWarnings("unchecked")
@Test
public void testDynamicIterate() throws Exception {
    // Refer to CompositeKeyTest.testCompositeKeySelect()
    EmpVO vo = new EmpVO();
    // 7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500,30
    // --> EMP which the mgr is 7698
    // 7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600 --> O
    // 7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250 --> O
    // 7844,'TURNER','SALESMAN',7698,'1981-09-08',1500 --> O
    // 7900,'JAMES','CLERK',7698,'1981-12-03',950 --> X
    vo.setEmpNo(new BigDecimal(7521));

    // select
    EmpIncludesEmpListVO resultVO =
        empDAO.selectEmpIncludesEmpList(
            "selectEmpIncludesSameMgrMoreSalaryEmpList", vo);

    // check
    assertNotNull(resultVO);
    assertEquals(new BigDecimal(7521), resultVO.getEmpNo());
    assertEquals("WARD", resultVO.getEmpName());
    assertTrue(resultVO.getEmpList() instanceof List);
    assertEquals(3, resultVO.getEmpList().size());
    assertEquals(new BigDecimal(7499), resultVO.getEmpList().get(0)
        .getEmpNo());
    assertEquals(new BigDecimal(1600), resultVO.getEmpList().get(0)
        .getSal());
    assertEquals(new BigDecimal(7844), resultVO.getEmpList().get(1)
        .getEmpNo());
    assertEquals(new BigDecimal(1500), resultVO.getEmpList().get(1)
        .getSal());
    assertEquals(new BigDecimal(7654), resultVO.getEmpList().get(2)
        .getEmpNo());
    assertEquals(new BigDecimal(1250), resultVO.getEmpList().get(2)
        .getSal());

    // select
    List<JobHistVO> resultList =
        jobHistDAO.getSqlMapClientTemplate().queryForList(
            "selectJobHistListUsingDynamicIterate", resultVO);
```

```
        assertNotNull(resultList);
        // the jobhist of 7499, 7654, 7844 is one each according to the initial data
        assertEquals(3, resultList.size());

        assertEquals(new BigDecimal(7499), resultList.get(0).getEmpNo());
        assertEquals(new BigDecimal(7654), resultList.get(1).getEmpNo());
        assertEquals(new BigDecimal(7844), resultList.get(2).getEmpNo());

        SimpleDateFormat sdf =
            new SimpleDateFormat("yyyy-MM-dd", java.util.Locale.getDefault());
        assertEquals(sdf.parse("1981-02-20"), resultList.get(0).getStartDate());
        assertEquals(sdf.parse("1981-09-28"), resultList.get(1).getStartDate());
        assertEquals(sdf.parse("1981-09-08"), resultList.get(2).getStartDate());

    }
```

Execute the query of CompositeKeyTest created earlier for complicated object and execute the query for iterate test with parameter object.

Comparison when iterate property is parameter object vs. parameter object itself

```
        ..

        <!-- parameterClass is not indicated. Test by separating the case when collection key is transferred to map or immediately list.-->
        <!—If iterate element is used as an input parameter binding variable for search condition, use like #collection[]# -->
        <select id="selectDynamicIterateSimple" resultClass="egovMap">
                select
                        <isPropertyAvailable property="collection">
                                <iterate property="collection" conjunction=", ">
                                        '$collection[]$' as $collection[]$
                                </iterate>
                        </isPropertyAvailable>
                        <!—If delivering List immediately -->
                        <isNotPropertyAvailable property="collection">
                                <iterate conjunction=", ">
                                        '$[]$' as $[]$
                                </iterate>
                        </isNotPropertyAvailable>
                from dual
        </select>
```

Depending on whether collection property is included, iterate target is the example of iterate processing for the property of parameter object or parameter object itself. The area created as $property name$ is not processed as the bind variable of prepared statement like #property name# but the SQL statement itself is replaced.

Sample TestCase

```
..
    @SuppressWarnings("unchecked")
    @Test
    public void testDynamicIterateSimple() throws Exception {
        // As much as object size of Collection type
        List iterateList = new ArrayList();
        iterateList.add("a");
        iterateList.add("b");
        iterateList.add("c");

        // select
        Map resultMap =
```

```
            (Map) jobHistDAO.getSqlMapClientTemplate().queryForObject(
                "selectDynamicIterateSimple", iterateList);

        // check
        assertNotNull(resultMap);
        assertEquals("a", resultMap.get("a"));
        assertEquals("b", resultMap.get("b"));
        assertEquals("c", resultMap.get("c"));
        assertTrue(!resultMap.containsKey("d"));

        // Case of entering list to map with property of collection
        Map map = new HashMap();
        map.put("collection", iterateList);

        // select
        resultMap =
            (Map) jobHistDAO.getSqlMapClientTemplate().queryForObject(
                "selectDynamicIterateSimple", map);

        // check
        assertNotNull(resultMap);
        assertEquals("a", resultMap.get("a"));
        assertEquals("b", resultMap.get("b"));
        assertEquals("c", resultMap.get("c"));
        assertTrue(!resultMap.containsKey("d"));

        // we have tried Map, Set, Iterator as test for iterate, but following error occurs. (it seems that
it should be access to index like List or Array)
        // The 'xxx'(ex. collection) property of the XXX (ex. java.util.HashMap$EntryIterator) class is
not a List or
        // Array.
    }
```

From above, there is a difference in iterate tag processing between the first case of deliverying the parameter object itself to list and the case of delivering the key of ‘collection in parameter map to the list.

Nested iterate Calculation

Refer to example of sample sql mapping xml below.

Sample iterate calculation

```
        ..

        <select id="selectJobHistListUsingDynamicNestedIterate" parameterClass="map"
resultClass="jobHistVO">
                <![CDATA[
                        select EMP_NO       as empNo,
                               START_DATE as startDate,
                               END_DATE    as endDate,
                               JOB          as job,
                               SAL          as sal,
                               COMM         as comm,
                               DEPT_NO      as deptNo
                        from    JOBHIST
                ]]>
                <dynamic prepend="where">
                        <iterate property="condition" open="(" conjunction="and" close=")">
                                $condition[].columnName$ $condition[].columnOperation$
                                <isEqual property="condition[].nested" compareValue="true">
```

```
                                        <iterate property="condition[].columnValue" open="("
conjunction="," close=")">
                                                #condition[].columnValue[]#
                                        </iterate>
                                </isEqual>
                                <isNotEqual property="condition[].nested" compareValue="true">
                                        #condition[].columnValue#
                                </isNotEqual>
                        </iterate>
                </dynamic>
                        order by EMP_NO, START_DATE
        </select>
```

As a example of complex condition processing, general comparison operation and Nested Iterate processing are used at the same time. For query calling, columnName, columnOperation, columnValue are transferred to multi; columnName and columnOperation processed as direct replaced Text in sql statement; columnValue as bind variable. At this time, if the additionally set value is true, determine that columnValue is the in consition statement and process as nested iterate. Following test case dynamically adds the following condition statement depending on parameter object setting.

where (DEPT_NO = ? and SAL < ? and JOB in (? , ?))

Sample TestCase

```
..
    @SuppressWarnings("unchecked")
    @Test
    public void testDynamicNestedIterate() throws Exception {
        // nested iterate tag test – add the list containing columnName, columnOperation,
columnValue in the form of Map  with condition as a key and add to parameter map
        // If columnValue should be solved with nested iterate(ex. in condition statement) set
additionally with nested 'true' for calling.
        Map map = new HashMap();
        List condition = new ArrayList();
        Map columnMap1 = new HashMap();
        columnMap1.put("columnName", "DEPT_NO");
        columnMap1.put("columnOperation", "=");
        columnMap1.put("columnValue", new BigDecimal(30));
        condition.add(columnMap1);

        Map columnMap2 = new HashMap();
        columnMap2.put("columnName", "SAL");
        columnMap2.put("columnOperation", "<");
        columnMap2.put("columnValue", new BigDecimal(3000));
        condition.add(columnMap2);

        Map columnMap3 = new HashMap();
        columnMap3.put("columnName", "JOB");
        columnMap3.put("columnOperation", "in");
        List jobList = new ArrayList();
        jobList.add("CLERK");
        jobList.add("SALESMAN");
        columnMap3.put("columnValue", jobList);
        // Inform that List is included as nested with flag
        columnMap3.put("nested", "true");
        condition.add(columnMap3);

        map.put("condition", condition);

        // select
        List<JobHistVO> resultList =
            jobHistDAO.getSqlMapClientTemplate().queryForList(
```

```
                "selectJobHistListUsingDynamicNestedIterate", map);

        // check
        assertNotNull(resultList);

        // Result data
        // Empno Startdate Enddate Job Sal Comm Deptno
        // 1 7499 81/02/20 SALESMAN 1600 300 30
        // 2 7521 81/02/22 SALESMAN 1250 500 30
        // 3 7654 81/09/28 SALESMAN 1250 1400 30
        // cf.) 7698 81/05/01 MANAGER 2850 30 data is filtered in nested since there is no JOB with
'MANAGER' in conditino.
        // 4 7844 81/09/08 SALESMAN 1500 0 30
        // 5 7900 83/01/15 CLERK 950 30
        assertEquals(5, resultList.size());
        assertEquals(new BigDecimal(7499), resultList.get(0).getEmpNo());
        assertEquals(new BigDecimal(7521), resultList.get(1).getEmpNo());
        assertEquals(new BigDecimal(7654), resultList.get(2).getEmpNo());
        assertEquals(new BigDecimal(7844), resultList.get(3).getEmpNo());
        assertEquals(new BigDecimal(7900), resultList.get(4).getEmpNo());

        SimpleDateFormat sdf =
            new SimpleDateFormat("yyyy-MM-dd", java.util.Locale.getDefault());
        assertEquals(sdf.parse("1981-02-20"), resultList.get(0).getStartDate());
        assertEquals(sdf.parse("1981-02-22"), resultList.get(1).getStartDate());
        assertEquals(sdf.parse("1981-09-28"), resultList.get(2).getStartDate());
        assertEquals(sdf.parse("1981-09-08"), resultList.get(3).getStartDate());
        assertEquals(sdf.parse("1983-01-15"), resultList.get(4).getStartDate());

    }
```

As examined above, very complicated condition can be processed using Dynamic elements of iBATIS. However, if the condition processing is complicated, it is difficult to identify dynamic tag area and it is difficult to perfectly cope with these complicated requests with the tag of simple logic/arithmetic operation level. Later iBATIS version is expected to enable more flexible and dynamic processing.