Data Type

Appropriate use and related processing on the data type is very important in writing applications. In particular, it is required to use related mapping jdbc type supported by DBMS and type at Java application when saving and inquiring the data using database. Refer to use guide of general Data Type focusing on use example of suitable mapping of idbcType of specific DBMS and javaType in iBATIS environment.

Using Basic Data Type

iBATIS SQL Mapper framework processes binding and mapping on the parameter/result object based on the data type mapping on the table column of each DBMS supported by java type and JDBC driver on each attribute such as standard JavaBeans object (or maps) in the Java application area. When the jdbcType mapped on each javaType uses general AnsiSQL, the below gives comprehensive picture. Additional supported/unsupported jdbcType can vary according to certain DBMS vendor. Even if it uses the same jdbcType, the boundary max./min. value can be different according to types.

Below is the examples of using data types in iBATIS environment through basic insert/select on various primitive types, number types, text types, and data types.

Sample Type VO

```
public class TypeTestVO implements Serializable {

    private static final long serialVersionUID = -3653247402772333834L;

    private int id;

    private BigDecimal bigdecimalType;

    private boolean booleanType;

    private byte byteType;

    private String charType;

    private double doubleType;

    private float floatType;

    private int intType;

    private long longType;

    private short shortType;

    private String stringType;

    private Date dateType;

    private java.sql.Date sqlDateType;

    private Time sqlTimeType;

    private Timestamp sqlTimestampType;

    private Calendar calendarType;

    public int getId() {
        return id;
    }
```

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

public BigDecimal getBigdecimalType() {
	return bigdecimalType;
}

public void setBigdecimalType(BigDecimal bigdecimalType) {
	this.bigdecimalType = bigdecimalType;
}

public boolean isBooleanType() {
	return booleanType;
}

public void setBooleanType(boolean booleanType) {
	this.booleanType = booleanType;
}

public byte getByteType() {
	return byteType;
}

public void setByteType(byte byteType) {
	this.byteType = byteType;
}

public String getCharType() {
	return charType;
}

public void setCharType(String charType) {
	this.charType = charType;
}

public double getDoubleType() {
	return doubleType;
}

public void setDoubleType(double doubleType) {
	this.doubleType = doubleType;
}

public float getFloatType() {
	return floatType;
}

public void setFloatType(float floatType) {
	this.floatType = floatType;
}

public int getIntType() {
	return intType;
}

public void setIntType(int intType) {
	this.intType = intType;
}

public long getLongType() {
	return longType;
}
```

```
	public void setLongType(long longType) {
		this.longType = longType;
	}

	public short getShortType() {
		return shortType;
	}

	public void setShortType(short shortType) {
		this.shortType = shortType;
	}

	public String getStringType() {
		return stringType;
	}

	public void setStringType(String stringType) {
		this.stringType = stringType;
	}

	public Date getDateType() {
		return dateType;
	}

	public void setDateType(Date dateType) {
		this.dateType = dateType;
	}

	public java.sql.Date getSqlDateType() {
		return sqlDateType;
	}

	public void setSqlDateType(java.sql.Date sqlDateType) {
		this.sqlDateType = sqlDateType;
	}

	public Time getSqlTimeType() {
		return sqlTimeType;
	}

	public void setSqlTimeType(Time sqlTimeType) {
		this.sqlTimeType = sqlTimeType;
	}

	public Timestamp getSqlTimestampType() {
		return sqlTimestampType;
	}

	public void setSqlTimestampType(Timestamp sqlTimestampType) {
		this.sqlTimestampType = sqlTimestampType;
	}

	public Calendar getCalendarType() {
		return calendarType;
	}

	public void setCalendarType(Calendar calendarType) {
		this.calendarType = calendarType;
	}

}
```

Each attribute of above TypeTestVO is the sample. Let's look at the mapping jdbc type through DDL of each DBMS.

Sample TYPETEST Table Hsqldb DDL script

```
create table TYPETEST (
    id numeric(10,0) not null,
    bigdecimal_type numeric(19,2),
    boolean_type boolean,
    byte_type tinyint,
    char_type char(1),
    double_type double,
    float_type float,
    int_type integer,
    long_type bigint,
    short_type smallint,
    string_type varchar(255),
    date_type date,
    sql_date_type datetime,
    sql_time_type time,
    sql_timestamp_type timestamp,
    calendar_type timestamp,
    primary key (id)
);
```

It is the example of hsqldb of above create sql sentence and is the example that follows the standards for Data Type of Ansi SQL actually. boolean type is directly supported. It is known from the following test case that it is not reasonable to use various jdbcType such as tinyint, double, date and time.

Sample SQL Mapping XML

```
<sqlMap namespace="TypeTest">

        <typeAlias alias="typeTestVO"
                type="egovframework.rte.psl.dataaccess.vo.TypeTestVO" />

        <!-- CalendarTypeHandler is registered in sql-map-config.xml -->
        <typeAlias alias="calendarTypeHandler"
type="egovframework.rte.psl.dataaccess.typehandler.CalendarTypeHandler"/>

        <resultMap id="typeTestResult" class="typeTestVO">
                <result property="id" column="ID" />
                <result property="bigdecimalType" column="BIGDECIMAL_TYPE" />
                <result property="booleanType" column="BOOLEAN_TYPE" />
                <result property="byteType" column="BYTE_TYPE" />
                <result property="charType" column="CHAR_TYPE" />
                <result property="doubleType" column="DOUBLE_TYPE" />
                <result property="floatType" column="FLOAT_TYPE" />
                <result property="intType" column="INT_TYPE" />
                <result property="longType" column="LONG_TYPE" />
                <result property="shortType" column="SHORT_TYPE" />
                <result property="stringType" column="STRING_TYPE" />
                <result property="dateType" column="DATE_TYPE" />
                <result property="sqlDateType" column="SQL_DATE_TYPE" />
                <result property="sqlTimeType" column="SQL_TIME_TYPE" />
                <result property="sqlTimestampType" column="SQL_TIMESTAMP_TYPE" />
                <result property="calendarType" column="CALENDAR_TYPE"
typeHandler="calendarTypeHandler" />
        </resultMap>

        <insert id="insertTypeTest" parameterClass="typeTestVO">
                <![CDATA[
                        insert into TYPETEST
                                    (ID,
                                     BIGDECIMAL_TYPE,
                                     BOOLEAN_TYPE,
                                     BYTE_TYPE,
                                     CHAR_TYPE,
                                     DOUBLE_TYPE,
                                     FLOAT_TYPE,
                                     INT_TYPE,
                                     LONG_TYPE,
                                     SHORT_TYPE,
                                     STRING_TYPE,
                                     DATE_TYPE,
                                     SQL_DATE_TYPE,
                                     SQL_TIME_TYPE,
                                     SQL_TIMESTAMP_TYPE,
                                     CALENDAR_TYPE)
                        values       (#id#,
                                     #bigdecimalType#,
                                     #booleanType#,
                                     #byteType#,
                                     #charType:CHAR#,
                                     #doubleType#,
                                     #floatType#,
                                     #intType#,
                                     #longType#,
                                     #shortType#,
                                     #stringType#,
                                     #dateType#,
                                     #sqlDateType#,
                                     #sqlTimeType#,
                                     #sqlTimestampType#,
                                     #calendarType,handler=calendarTypeHandler#)
                ]]>
        </insert>

        <select id="selectTypeTest" parameterClass="typeTestVO"
                resultMap="typeTestResult">
                <![CDATA[
                        select ID,
                               BIGDECIMAL_TYPE,
                               BOOLEAN_TYPE,
                               BYTE_TYPE,
                               CHAR_TYPE,
                               DOUBLE_TYPE,
                               FLOAT_TYPE,
                               INT_TYPE,
                               LONG_TYPE,
                               SHORT_TYPE,
                               STRING_TYPE,
                               DATE_TYPE,
                               SQL_DATE_TYPE,
                               SQL_TIME_TYPE,
                               SQL_TIMESTAMP_TYPE,
                               CALENDAR_TYPE
                        from    TYPETEST
                        where   ID = #id#
                ]]>
        </select>

</sqlMap>
```

It is sql mapping xml processing insert/select through TypeTestVO JavaBeans object. Select result object mapping is processed by defining resultMap; parameter binding of input and search condition is processed through Inline Parameter method.

resultMap or parameterMap(Inline Parameter too) can indicate for java/jdbc type explicitly like javaType="string", jdbcType="VARCHAR". (recommended in terms of performance, however, the type instruction that does not match with the reality may cause error at runtime) In addition, at above Inline parameter processing, as may be confirmed from instruction with handler=calendarTypeHandler for calendar property and from the instruction with typeHandler="calendarTypeHandler" at the time of resultMap processing, the user implements typeHandler for the area that cannot be covered with general java-jdbc mapping, and provides the logic processing for type conversion so that it can be expanded like calendar type ↔ TIMESTAMP conversion as shown above.

In above, TypeTestVO and SQL Mapping XML are reused without change at the following additional DBMS test, and nonsupport of some Data Type/ use mapping type per DBMS/boundary value difference, avoid the problematic area through avoidance or some logic branch in DDL / TestCase and refer to this since it was tested for reuse from overall aspect.

Sample TestCase

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath*:META-INF/spring/context-*.xml" })
@TransactionConfiguration(transactionManager = "txManager", defaultRollback = false)
@Transactional
public class DataTypeTest extends TestBase {

    @Resource(name = "typeTestDAO")
    TypeTestDAO typeTestDAO;

    @Before
    public void onSetUp() throws Exception {

        // Initialize DB from sql file externally (delete/create TypeTest existing table)
        SimpleJdbcTestUtils.executeSqlScript(
            new SimpleJdbcTemplate(dataSource), new ClassPathResource(
                "META-INF/testdata/sample_schema_ddl_typetest_" + usingDBMS
                    + ".sql"), true);
    }

    public TypeTestVO makeVO() throws Exception {
        TypeTestVO vo = new TypeTestVO();
        vo.setId(1);
        vo.setBigdecimalType(new BigDecimal("99999999999999999.99"));
        vo.setBooleanType(true);
        vo.setByteType((byte) 127);
        // Declared as String at VO. To use as char, required to write TypeHandler
        vo.setCharType("A");
        // From Oracle 10g, double precision type fails to accept Double.MAX_VALUE.
        // From oracle jdbc driver, deliver Double.MAX_VALUE and Overflow Exception trying to bind 
1.7976931348623157E308 error occurs
        // If testing at mysql 5.0, Double.MAX_VALUE input is possible, but returned to Infinity at the 
time of inquiry
        // Exception occurs at the time of tibero - Double.MAX_VALUE input
        vo.setDoubleType(isHsql ? Double.MAX_VALUE : 1.7976931348623157d);
        // If testing at mysql 5.0, cannot enter Float.MAX_VALUE
        vo.setFloatType(isMysql ? (float) 3.40282 : Float.MAX_VALUE);
        vo.setIntType(Integer.MAX_VALUE);
        vo.setLongType(Long.MAX_VALUE);
        vo.setShortType(Short.MAX_VALUE);
        vo.setStringType("abcd 가나다라あいうえおｶｷｸｹｺ");
        SimpleDateFormat sdf =
            new SimpleDateFormat("yyyy-MM-dd", java.util.Locale.getDefault());
        vo.setDateType(sdf.parse("2009-02-18"));
        long currentTime = new java.util.Date().getTime();
        vo.setSqlDateType(new java.sql.Date(currentTime));
        vo.setSqlTimeType(new java.sql.Time(currentTime));
        vo.setSqlTimestampType(new java.sql.Timestamp(currentTime));
        vo.setCalendarType(Calendar.getInstance());

        return vo;
    }

    public void checkResult(TypeTestVO vo, TypeTestVO resultVO) {
        assertNotNull(resultVO);
        assertEquals(vo.getId(), resultVO.getId());
        assertEquals(vo.getBigdecimalType(), resultVO.getBigdecimalType());
        assertEquals(vo.getByteType(), resultVO.getByteType());
        // In case of mysql, note that it is inserted to current time if you enter null in timestamp
column
        if (vo.getCalendarType() == null && isMysql) {
            assertNotNull(resultVO.getCalendarType());
            // In case of mysql, 3 digit precision falls compared to timestamp of java
        } else if (vo.getCalendarType() != null && isMysql) {
            String orgSeconds =
                Long.toString(vo.getCalendarType().getTime().getTime());
            String mysqlSeconds =
                Long.toString(resultVO.getCalendarType().getTime().getTime());
            assertEquals(orgSeconds.substring(0, orgSeconds.length() - 3),
                mysqlSeconds.substring(0, mysqlSeconds.length() - 3));
        } else {
            assertEquals(vo.getCalendarType(), resultVO.getCalendarType());
        }
        assertEquals(vo.getCharType(), resultVO.getCharType());
        assertEquals(vo.getDateType(), resultVO.getDateType());
        // give delta for double as 1e-15.
        assertEquals(vo.getDoubleType(), resultVO.getDoubleType(), isMysql
            ? 1e-14 : 1e-15);
        // give delta for float as 1e-7.
        assertEquals(vo.getFloatType(), resultVO.getFloatType(), 1e-7);
        assertEquals(vo.getIntType(), resultVO.getIntType());
        assertEquals(vo.getLongType(), resultVO.getLongType());
        assertEquals(vo.getShortType(), resultVO.getShortType());
        // comparison of date in case of java.sql.Date
        if (vo.getSqlDateType() != null) {
            assertEquals(vo.getSqlDateType().toString(), resultVO
                .getSqlDateType().toString());
        }

        // in case of mysql, note that current time is inserted when enter null in timestamp column
        if (vo.getSqlTimestampType() == null && isMysql) {
            assertNotNull(resultVO.getSqlTimestampType());
        } else if (vo.getCalendarType() != null && isMysql) {
            String orgSeconds =
                Long.toString(vo.getSqlTimestampType().getTime());
            String mysqlSeconds =
                Long.toString(resultVO.getSqlTimestampType().getTime());
            assertEquals(orgSeconds.substring(0, orgSeconds.length() - 3),
                mysqlSeconds.substring(0, mysqlSeconds.length() - 3));
        } else {
            assertEquals(vo.getSqlTimestampType(), resultVO
                .getSqlTimestampType());
        }
        // in case of java.sql.Time, compare time only
        if ((isHsql || isOracle || isTibero || isMysql)
            && vo.getSqlTimeType() != null) {
```

```
            assertEquals(vo.getSqlTimeType().toString(), resultVO
                .getSqlTimeType().toString());
        } else {
            assertEquals(vo.getSqlTimeType(), resultVO.getSqlTimeType());
        }
        assertEquals(vo.getStringType(), resultVO.getStringType());
        assertEquals(vo.isBooleanType(), resultVO.isBooleanType());

    }

    @Test
    public void testDataTypeTest() throws Exception {
        // value is not set but inserted - id is 0 according to initial value of int
        TypeTestVO vo = new TypeTestVO();

        // insert
        typeTestDAO.insertTypeTest("insertTypeTest", vo);

        // select
        TypeTestVO resultVO = typeTestDAO.selectTypeTest("selectTypeTest", vo);

        // check
        checkResult(vo, resultVO);

        try {
            // duplication test
            typeTestDAO.insertTypeTest("insertTypeTest", vo);

            fail("key value duplicate error should occur.");
        } catch (Exception e) {
            assertNotNull(e);
            assertTrue(e instanceof DataIntegrityViolationException);
            assertTrue(e.getCause() instanceof SQLException);
        }

        // Enter and re-inquire DataType test data
        vo = makeVO();

        // insert
        typeTestDAO.insertTypeTest("insertTypeTest", vo);

        // select
        resultVO = typeTestDAO.selectTypeTest("selectTypeTest", vo);

        // check
        checkResult(vo, resultVO);

    }

}
```

In above, check the processing for input/view by setting the test data (value in meaningful use example or boundary value) in each property or whether DataIntegrityViolationException of spring occurs if key value duplicates, input/view without value in each property of TypeTestVO, and check the example of type mapping of java ↔ DBMS. In particular, above makeVO method can check that max value of boundary value may differ depending on the type of DBMS for specific javaType and checkResult can check that there is a difference in precision (when entering, in Date of java , data or time information is restricted or precision of second level is low compared to java, if viewing the result of jdbcType for input value javaType, expressed in details beyond year, month, date, hour, minute and second) supported or the initial value when entering null depending on DBMS with regard to data processing type in particular. General mapping for java-jdbc type may be suitable if you understand the example of above Hsqldb. Through DDL example of specific DBMS below, it can be a reference for use of Data Type in each database environment.

Sample TYPETEST Table Oracle (10gR2 standard test) DDL script

```
create table TYPETEST (
    id number(10,0) not null,
    bigdecimal_type number(19,2),
    boolean_type number(1,0),
    byte_type number(3,0),
    char_type char(1),
    double_type double precision,
    float_type float,
    int_type number(10,0),
    long_type number(19,0),
    short_type number(5,0),
    string_type varchar2(255),
    date_type date,
    sql_date_type date,
    sql_time_type timestamp,
    sql_timestamp_type timestamp,
    calendar_type timestamp,
    primary key (id)
);
```

Sample TYPETEST Table Mysql (5.X) DDL script

```
create table TYPETEST (
    id numeric(10,0) not null,
    bigdecimal_type numeric(19,2),
    boolean_type boolean,
    byte_type tinyint,
    char_type char(1),
    double_type double,
    float_type float,
    int_type integer,
    long_type bigint,
    short_type smallint,
    string_type varchar(255),
    date_type date,
    sql_date_type datetime,
    sql_time_type time,
    sql_timestamp_type timestamp,
    calendar_type timestamp,
    primary key (id)
);
```

Sample TYPETEST Table Tibero(3.x) DDL script

```
create table TYPETEST (
    id numeric(10,0) not null,
    bigdecimal_type number(19,2),
    boolean_type number(1),
    byte_type number(3),
    char_type char(1),
    double_type double precision,
    float_type float,
    int_type integer,
    long_type integer, /* integer includes the scope of bigint */
    short_type smallint,
    string_type varchar(255),
    date_type date,
    sql_date_type date,
    sql_time_type date,     /* generate problems in configuring as time, timestamp */
    sql_timestamp_type timestamp,
    calendar_type timestamp,
    primary key (id)
);
```