# Dynamic SQL

When using MyBatis, HotRod leverages the MyBatis dynamic SQL capabilities to execute SQL statements that include or exclude fragments of SQL at runtime based on the parameter values provided by the application on each execution.

While—accurately described on the MyBatis manual—“*working with Dynamic SQL will never be a party*”, the dynamic SQL strategy reduces considerably the amount of work the developer needs to do when writing heavily parameterized SQL.

The example and description shown in this section uses a SQL <update> tag. However, all this functionality is valid as well for the <select> tags

For details on the MyBatis dynamic syntax for SQL statements please see the MyBatis documentation. The one-sentence description is that MyBatis enhances the SQL definitions by allowing <if>, <choose>, <when>, <otherwise>, <trim>, <where>, <set>, and <foreach> tags to be added to it to transform and reshape the SQL sentence at runtime.

The example below shows a SQL update whose structure changes dynamically:

<update java-method-name="addFreebiesToOrders">

<![CDATA[

update purchase\_order

<set>

<if test="#{discount,javaType=java.lang.Double,jdbcType=NUMERIC} != null">

discount = #{discount}

,</if>

<if test="#{extraQuantity,javaType=java.lang.Long,jdbcType=NUMERIC} != null">

quantity = quantity + #{extraQuantity}

,</if>

<if test="#{shippingType,javaType=java.lang.Integer,jdbcType=NUMERIC} != null">

shipping\_type = #{shippingType}

,</if>

<if test="#{freeWarranty,javaType=java.lang.Integer,jdbcType=NUMERIC} != null">

free\_warranty = #{freeWarranty}

,</if>

</set>

<where>

<if test="#{widgetType,javaType=java.lang.Integer,jdbcType=NUMERIC} != null">

widget\_type = #{widgetType}

</if>

<if test="#{zipCode,javaType=java.lang.String,jdbcType=VARCHAR} != null">

AND zip\_code = #{zipCode}

</if>

<if test="#{orderDate,javaType=java.sql.Date,jdbcType=DATE} != null">

AND order\_date &lt; #{orderDate}

</if>

</where>

]]>

</update>

This tag produces the static DAO java method:

public static int addFreebiesToOrders(

java.lang.Double discount,

java.lang.Long extraQuantity,

java.lang.Integer shippingType,

java.lang.Integer freeWarranty,

java.lang.Integer widgetType,

java.lang.String zipCode,

java.sql.Date orderDate) { ... }

In the example above, depending on which parameters are null or not, one or more freebies (a discount, extra quantity, better shipping, and/or free warranty) are given to the purchase orders. Also, the selection criteria can be any combination of widget type, zip code, and/or before a certain order date.

In sum, there are 16 possible combinations of freebies, and there are 8 possible combinations for the selection criteria. The full combination of cases adds up to 128 different cases (16 × 8). That is, if you decided not to use dynamic SQL you would need to write 128 different SQL updates to provide the same functionality this single dynamic SQL update offers. In cases like this one is where dynamic SQL shines.

Of course, it's not trivial to write the above dynamic SQL statement, but it's much less work than the alternative.

**A Few Notes on the Dynamic SQL Syntax**

Please note the CDATA section is needed for dynamic SQL. Otherwise the HotRod parser would try to interpret the dynamic tags as part of the HotRod configuration file.

In the CDATA section, some reserved XML characters found outside the tag definitions—i.e. in the SQL statement per se—must be escaped as an XML entity. That is:

* any < must be escaped as &lt;
* any > character must be escaped as &gt;
* any & character must be escaped as &amp;

For example, the date comparison shows the character < being escaped. Let me repeat this once more: the character escaping only affects the SQL code and not the dynamic tags themselves.

As in any non-dynamic SQL <update> tag the first time a parameter is used—regardless if it occurs inside a tag or in the SQL code—its full definition is required. All subsequent occurrences of the parameter use the simple form. For example, the first time the parameter discount is used it takes the full form:

#{discount,javaType=java.lang.Double,jdbcType=NUMERIC}

Then, when used again it takes the simplified form:

#{discount}

Finally, the dynamic SQL includes <set>, <if>, and <where> tags. These tags seem intuitive enough to use but they actually come with nice extras. If you re-read the example carefully, you may notice the commas that separate the set sections in the update are taken care of automatically by the <set> tag in all—I repeat ALL—parameter combinations. Also, the presence or absence of the AND SQL word is also managed automatically for you by the <where> tag. See the MyBatis manual for details on this.