# Column Configuration Reference

Columns are a central concept used in the DAO generation. Essentially all data sets retrieved from the database have a tabular form. When the data sets are retrieved each row is gathered as a single DAO (that is the DAO acting as a VO) and each column value for the row is placed in a DAO property.

In the simplest form—when directly dumping the content of a table—each column corresponds to a column on the related table. For example, when using a SQL query such as:

select \* from vehicle

This SQL query will produce a tabular data with the exact columns the originating table VEHICLE has.

However, when producing more elaborated SQL queries the resulting tabular data can have little resemblance to the originating database table. The following SQL query—based on the same table—returns a totally dissimilar tabular structure:

select avg(mileage/age) as mpy, age

from vehicle grouped by age

In this case the resulting tabular structure has two columns and bares little resemblance to the VEHICLE table.

Columns are used by HotRod in the following cases:

* For the basic CRUD operations of a database table. In this case each column has a one-to-one relationship with the related table. Each column produces a DAO property.
* For read operations on a database view. In this case each column has a one-to-one relationship with the view column.
* When running arbitrary SQL selects. In this case each column corresponds to a custom column from the resulting tabular data produced by the SQL select.

Now, why are we delving into all these details? Well, it happens that while most columns are very straightforward to interpret and use, other ones can be quite tricky to deal with.

Usually HotRod produces quite appropriate Java names and Java types for each column. However, in some cases the default names and/or types may not be the optimal ones according to the developer's expectations or needs. For example, the developer may consider some columns may be better interpreted by a different Java type (as a java.lang.Double, rather than as a java.math.BigDecimal) based on extra business knowledge that is entirely out of scope for HotRod.

In other words, the default attributes HotRod provides in some cases may not be the exact ones the developer wants. To overcome this obstacle the configuration file allows the developer to override default values using the <column> tag.

To find out how the <column> tag affects the <table> tag see the Configuration Reference section for the Tables.

To find out how the <column> tag affects the <view> tag see the Configuration Reference section for the Views.

To find out how the <column> tag affects the <select> tag see the Configuration Reference section for the Selects.

## The <column> tag

In most cases the metadata retrieved from the database is suitable enough to perform the whole code generation. In these cases there's nothing the developer will be interested on tweaking for the resulting tabular data.

There are, however, cases where the developer could change some settings. Typical cases are:

* the column name of an old or legacy column is cryptic, so the developer wants to use a nicer/clearer name for the DAO property. Or maybe there's a misspelling; yes, it happens.
* the default DAO property type for a column is not the one the developer prefers, and he/she wants to specify a different one.
* the column has an obscure database type and HotRod does not offer any default Java property type for it. A <column> tag is necessary if the column needs to be used.

The <column> tag attributes are:

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Required** |
| name | The name of the table column. | Yes |
| java-name | The Java name for the DAO property corresponding to this column. If not specified, the code generator produces a Java name based on the table column name. | No |
| java-type | The Java type for the DAO property corresponding to this column. If not specified, the code generator produces a Java type according to the specifics of the database data type. See the Supported Databases & Column Types section regarding your specific database for details on how the Java types are produced. | No |
| jdbc-type | Overrides the JDBC type provided by the database metadata. This is an advanced setting that you should provide only if you want to try how well different Java and/or JDBC types are handled by the JDBC driver when reading or writing data from and to the database. You probably would like to set this value only if you are having trouble with the JDBC driver, or when dealing with obscure database data types. Otherwise, stay away from this setting. | No |
| is-lob | Force the LOB flag for this column. Only use this value when dealing with obscure database data types HotRod fails to decide if they are or not LOB types. Valid values are true or false. Defaults to false. | No |
| initial-value | The initial value used on the version control column when inserting a new row. Must be an integer number, positive or negative. Defaults to zero. | No |
| max-value | The maximum value a version control column can reach when updating a row, before cycling down to the minimum value. Must be an integer number, positive or negative. | No |
| min-value | The minimum value a version control column is set to, after reaching its maximum value. Must be an integer number, positive or negative. | No |

The most common attributes that are tweaked are java-name (for oddly named columns) and java-type (for not clear cut column types).

The rest of the attributes are well worked out most of the time by the code generator, so there rarely any need to openly override their values.