# Support for Oracle database

The HotRod Oracle adapter automatically maps known database column types to DAO Java types. In most of the cases this default Java type is well suited to handle the database values. However, when needed the default Java type of a property can be overridden by a custom type if it's provided by the developer.

## Default Java Types

If a custom Java type is not specified HotRod will use the following rules to decide which Java type to use for each Oracle column. In yellow is the DAO property type. In parenthesis the actual object type returned by the Oracle JDBC driver, that on occasions may be different.

Please note that the Java types for the Oracle columns may vary depending on the specific version and variant of the RDBMS, the operating system where the database engine is running, and the JDBC driver version.

|  |  |
| --- | --- |
| **Oracle Column Type** | **Default Java Type** |
| NUMBER(p,s),  DECIMAL(p,s),  DEC(p,s),  NUMERIC(p,s),  NUM(p,s) | If neither p or s are specified:   * java.math.BigDecimal   If both are specified and s is different from zero the Java type is:   * java.math.BigDecimal   if p is specified and s is not specified or specified with a value of zero:   * if p <= 2: java.lang.Byte * if 2 < p <= 4: java.lang.Short * if 4 < p <= 9: java.lang.Integer * if 8 < p <= 18: java.lang.Long * if p > 18: java.math.BigInteger |
| SMALLINT,  INTEGER,  INT | java.math.BigInteger  **Note**: SMALLINT, INTEGER, and INT are equivalent to NUMBER(38). |
| FLOAT(p) | if p is not specified (i.e. a 126-bit float):   * java.math.BigDecimal   if p is specified:   * if p <= 23: java.lang.Float * if 24 <= p <= 52: java.lang.Double * if p >=53: java.math.BigDecimal |
| REAL | java.math.BigDecimal  **Note**: REAL is equivalent to FLOAT(63). |
| DOUBLE PRECISION | java.math.BigDecimal  **Note**: DOUBLE PRECISION is equivalent to FLOAT(126). |
| BINARY\_FLOAT | java.lang.Float |
| BINARY\_DOUBLE | java.lang.Double |
| CHAR(n),  VARCHAR(n),  VARCHAR2(n),  NCHAR(n),  NVARCHAR2(n) | java.lang.String |
| CLOB,  NCLOB | java.lang.String \* |
| LONG | No default java type \*\* |
| RAW(n),  LONG RAW | byte[] |
| BLOB | byte[] \* |
| BFILE | No default java type \*\* |
| DATE | java.util.Date |
| TIMESTAMP | java.sql.Timestamp |
| TIMESTAMP WITH TIME ZONE | java.sql.Timestamp |
| TIMESTAMP WITH LOCAL TIME ZONE | java.sql.Timestamp |
| ROWID | java.lang.String |
| UROWID | java.lang.Object (oracle.sql.ROWID) |
| XMLTYPE | No default java type \*\* |
| URITYPE | java.lang.Object (oracle.sql.STRUCT) |
| INTERVAL YEAR TO MONTH | java.lang.Object (oracle.sql.INTERVALYM) |
| INTERVAL DAY TO SECOND | java.lang.Object (oracle.sql.INTERVALDS) |
| VARRAY(n) | java.lang.Object (oracle.sql.ARRAY) |
| STRUCT | java.lang.Object (oracle.sql.STRUCT) |
| REF | java.lang.Object |

\* LOB types are by default read all at once into memory as byte arrays. They can also be read/written using streaming instead of loading them all at once. To do this you’ll need to write a custom MyBatis TypeHandler.

\*\* It may be possible to read/write from/to these columns using a MyBatis custom TypeHandler. This is, however, not an out-of-the-box solution.

## Custom Java Types

To override the default Java type see the reference section for the tables, views, and selects. The example Custom DAO Property Java Types shows a some cases where a custom type overrides the default type. To do it add a <column> tag in a <table>, <view>, or <select> definition as in:

<table name=*"my\_table"*>

<column name=*"price"* java-type=*"java.lang.Double"* jdbc-type=*"NUMERIC"* />

</table>

This configuration will force the property type to java.lang.Double instead of java.math.BigDecimal (the default type).