# Support for PostgreSQL database

The HotRod PostgreSQL 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 PostgreSQL column. In yellow is the DAO property type. In parenthesis the actual object type returned by the PostgreSQL JDBC driver, that on occasions may be different.

Please note that the Java types for the PostgreSQL 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.

|  |  |
| --- | --- |
| **PostgreSQL Column Type** | **Default Java Type** |
| SMALLINT,  INT2,  SMALLSERIAL | java.lang.Short |
| INTEGER,  INT,  INT4,  SERIAL | java.lang.Integer |
| BIGINT,  INT8,  BIGSERIAL | java.lang.Long |
| DECIMAL(p,s),  NUMERIC(p,s) | If neither p or s are specified:   * java.math.BigDecimal   If s is specified and different from zero the Java type is:   * java.math.BigDecimal   if 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 |
| REAL | java.lang.Float |
| DOUBLE PRECISION | java.lang.Double |
| MONEY | java.math.BigDecimal |
| CHAR(n),  CHARACTER(n),  VARCHAR(n),  CHARACTER VARYING(n), | java.lang.String |
| TEXT | java.lang.String |
| BYTEA | byte[] |
| DATE | java.sql.Date |
| TIMESTAMP(n),  TIMESTAMP(n) WITHOUT TIME ZONE,  TIMESTAMPTZ(n),  TIMESTAMP(n) WITH TIME ZONE | java.sql.Timestamp |
| TIME(n),  TIME(n) WITHOUT TIME ZONE,  TIMETZ(n),  TIME(n) WITH TIME ZONE | java.sql.Timestamp \* |
| BOOLEAN,  BOOL | java.lang.Boolean |
| INTERVAL <fields> (n) | No default HotRod data type |
| XML | No default HotRod data type |
| POINT,  LINE,  LSEG,  BOX,  PATH,  POLYGON,  CIRCLE | No default HotRod data type |
| CIDR,  INET,  MACADDR | No default HotRod data type |
| BIT(n),  BIT VARYING(n) | No default HotRod data type |
| UUID | java.lang.Object \*\* |
| JSON,  JSONB | No default HotRod data type |
| (arrays, such as)  INTEGER[],  CHAR[][],  INTEGER ARRAY | No default HotRod data type |
| INT4RANGE,  INT8RANGE,  NUMRANGE,  TSRANGE,  TSTZRANGE,  DATERANGE | No default HotRod data type |
| (enum data types) | No default HotRod data type |
| (composite types) | No default HotRod data type |
| OID,  REGPROC,  REGPROCEDURE,  REGOPER,  REGOPERATOR,  REGCLASS,  REGTYPE,  REGROLE,  REGNAMESPACE,  REGCONFIG,  REGDICTIONARY | No default HotRod data type |

\* In the special case of a precision of zero, a java.sql.Time type would be enough to store any time of the day without fractional seconds. However, since the majority of cases will have a different precision this type defaults to java.sql.Timestamp in all cases; this type can handle up to 9 decimal places.

\*\* Even though, the java.util.UUID type is able to save a value into the database, apparently it cannot read from the database into a Java program. Therefore, the java.lang.Object type is safer, but you'll need to cast it after retrieving a value.

## 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).