Denodo Hibernate Dialect User Manual

Revision 20231006

Denodo Inc.: 525 University Avenue, Suite 31, Palo Alto, CA 94301. USA www.denodo.com

CONTENTS

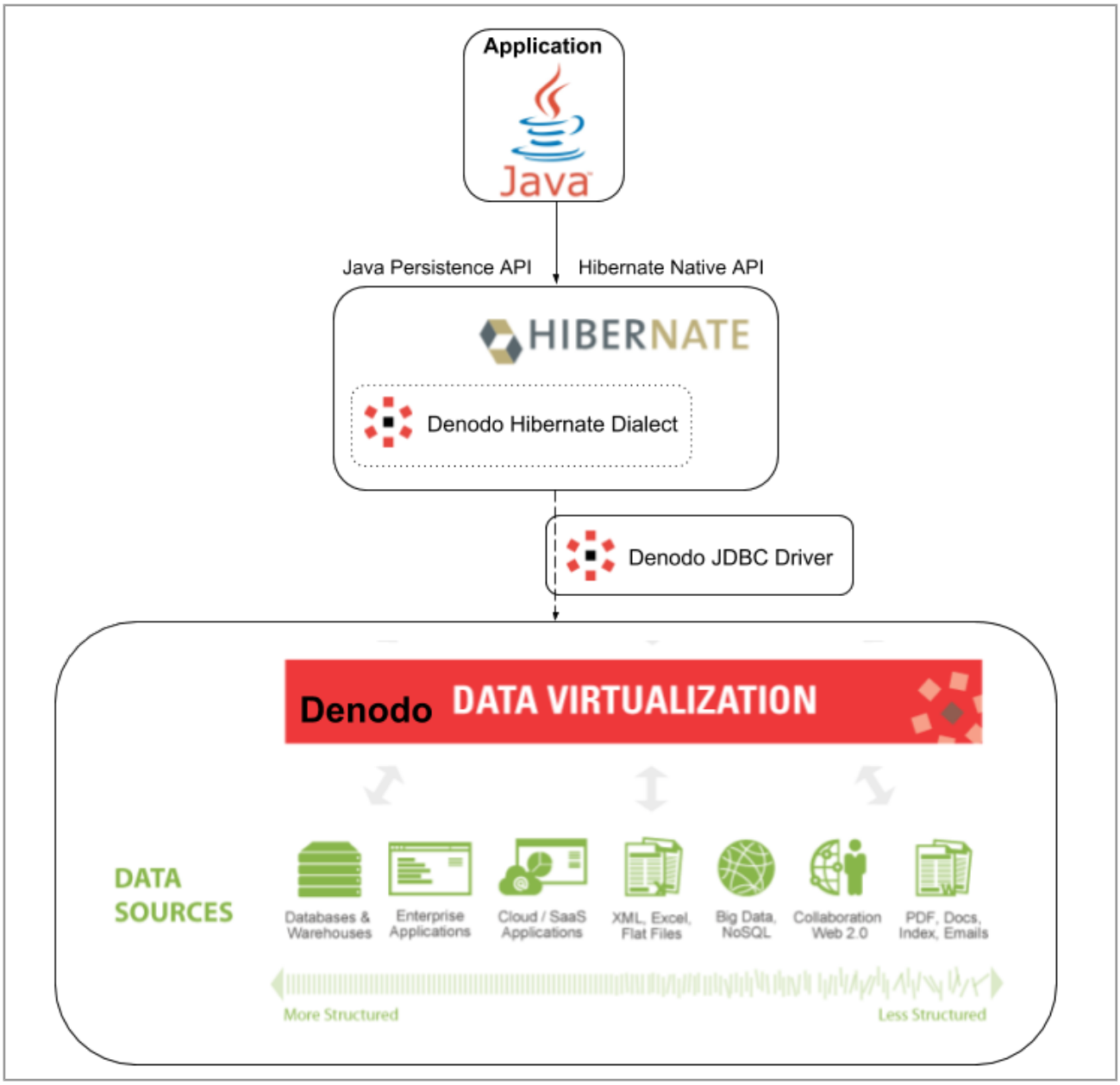
1 OVERVIEW

Hibernate is a framework that provides object/relational mapping (ORM) services for relational databases.

Hibernate is database agnostic. This means that it can work with different databases. However, as databases implement slight variations of the SQL standard, Hibernate includes vendor specific **dialects** to handle these differences and accomplishes tasks like obtaining a primary key identifier or structuring a SELECT query.

The **Denodo Hibernate Dialect** is the dialect provided by Denodo to generate the appropriate VQL so Hibernate can deal with Denodo databases.

This dialect supports Hibernate versions 6.1 to 6.3.



Hibernate - Denodo integration

Please note: Hibernate-based applications typically perform a number of write operations as a part of their execution, and some of these write operations might need to be performed in transactional contexts. Please note that the virtual nature of the Denodo Platform as a database sets some limits on what specific views can allow write operations, and what levels of transactionality can be achieved in each case depending on the nature, configuration and specific architecture of the involved data sources. Please refer to the official Denodo documentation for more information on writability and transactionality via JDBC on Denodo views.

2 JAVA CONFIGURATION

2.1 DEPENDENCIES

You will need to have the following libraries in your classpath:

- Hibernate
- Denodo VDP JDBC driver
- Denodo Hibernate Dialect jar: `denodo-hibernate-dialect-<version>.jar` in the `/dist` directory of the Denodo Hibernate Dialect distribution

2.2 PROPERTIES

You need to add the following property in your Hibernate configuration file:

```
dialect=com.denodo.connect.hibernate.dialect.DenodoDialect
```

besides the JDBC connection properties:

```
connection.driver_class=com.denodo.vdp.jdbc.Driver
connection.url=jdbc:vdb://localhost:9999/database
connection.username=…
connection.password=…
```

If yours is a Spring Boot application, configuration will look similar to this:

```
spring.datasource.driver-class-name=com.denodo.vdp.jdbc.Driver
spring.datasource.url=jdbc:vdb://localhost:9999/database
spring.datasource.username=…
spring.datasource.password=…
spring.jpa.properties.hibernate.dialect=com.denodo.connect.hibernate.dialect.DenodoDialect
#Avoids the appearance of HHH000424 in log during startup
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
```

2.3 ENTITY JAVA CLASS WITH AUTO-INCREMENT KEYS

If you need to use **primary key values that are automatically generated** by the underlying database by means of **auto-increment** columns, you can do so by using the identity sequence generation strategy in the Denodo Hibernate Dialect. This will only work for JDBC base views created on top of tables with an auto-increment column for their primary key.

Note this feature will only work from:

- ***Denodo 7.0 update 20201116.***
- ***Denodo 8.0 update 20210209.***

Your entity class will look similar to this:

```
package ...;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;

@Entity
public class Person {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int id;
   private String name;
   private int age;
   .............
}
```

2.4 ENTITY JAVA CLASS WITH ASSIGNED KEYS

If you do **not** need the **primary key value to be generated automatically** you only have to use the `@Id` annotation that marks which entity field is the primary key and initialize its value in your application.

```
package ...;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;

@Entity
public class Person {

   @Id
   private int id;
   private String name;
   private int age;
   .............
}
```

2.5 DATETIME MAPPINGS IN DENODO 7

To persist the new datetime types introduced in Denodo 7 you have to use the following Java types in you entity Java classes:

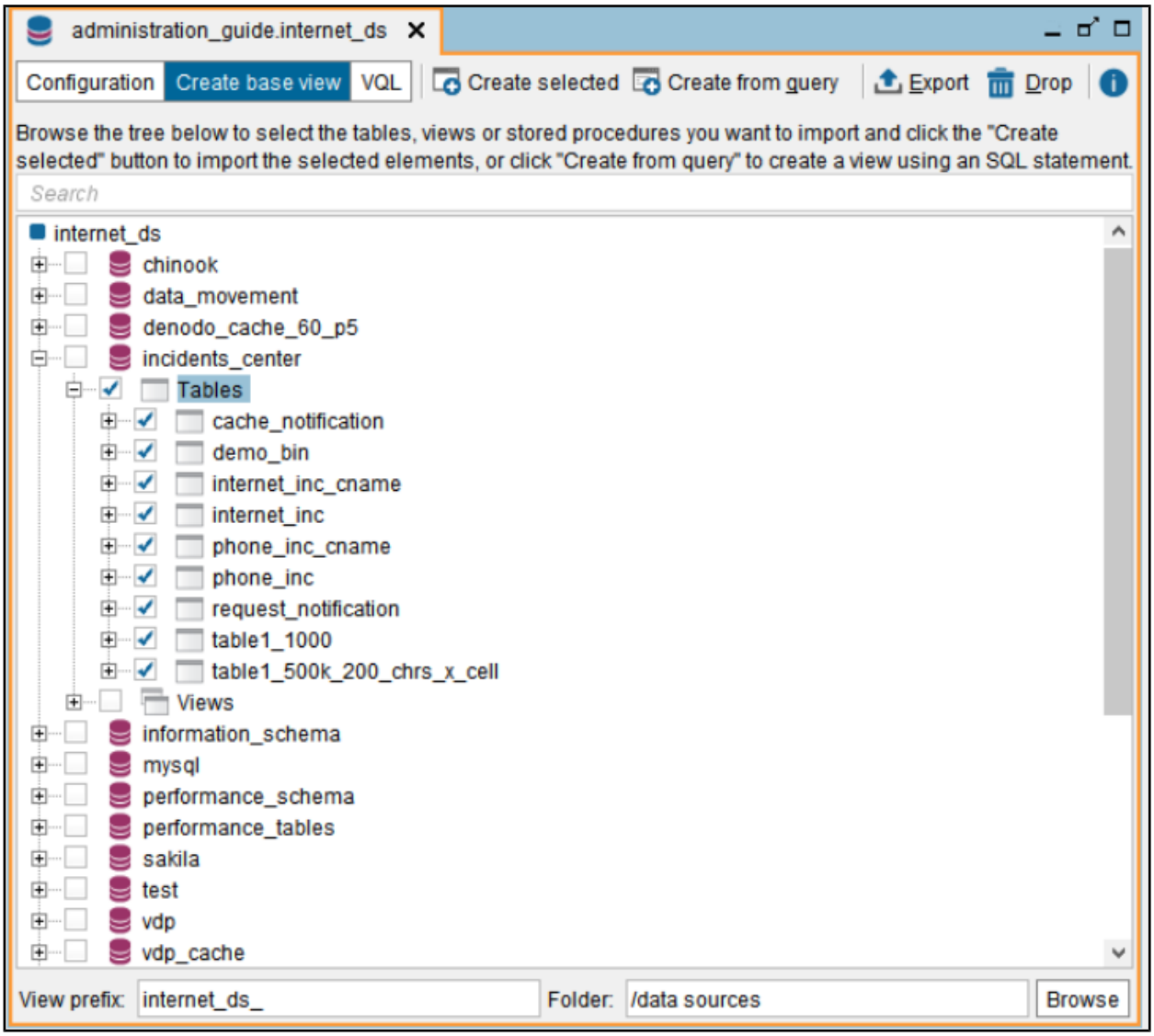
VDP Type	Java Type
`localdate`	`java.time.LocalDate` `java.sql.Date`
`time`	`java.time.LocalTime` `java.sql.Time`
`timestamp`	`java.time.LocalDateTime`
`timestamptz`	`java.time.OffsetDateTime` `java.sql.Timestamp`
`intervalyearmonth`	`java.time.Period`
`intervaldaysecond`	`java.time.Duration`

For more information on mappings between VDP datetime types and Java types, you can visit the Denodo 7 documentation: https://community.denodo.com/docs/html/browse/7.0/vdp/developer/access_through_jdbc/details_of_the_jdbc_interface/details_of_the_jdbc_interface#working-with-datetime-values-with-the-denodo-jdbc-driver

3 DENODO CONFIGURATION

3.1 GENERAL DENODO CONFIGURATION

You need to **create the JDBC data source and the base views** that correspond with your Java entities, using the Virtual DataPort Administration Tool.



4 LIMITATIONS

- Using `hibernate.hbm2ddl.auto` during application development is not supported because the `Denodo Hibernate Dialect` does not support DDL operations. All views mapped as objects in your Hibernate applications will need to exist in VDP before Hibernate can access them.

- Denodo supports auto-increment key generation through its JDBC driver for most database vendors typically used in Hibernate-based applications: Oracle, MySQL, PostgreSQL, SQL Server, Azure SQL, etc. but you might need to check the availability of this kind of support in Denodo VDP for your specific database if you are using a different vendor, or a database version that might not include such support out of the box. Also note that the auto-increment key generation strategy can only be used with JDBC-backed *base* views.

- Write support is restricted to those views (base or derived) that can be considered “updateable” by Denodo Virtual DataPort. Also, transaction capabilities are restricted to the types and levels that Denodo Virtual DataPort is able to offer depending on the specific views involved.