7. Database access7.1. ConnectionProvider As an ORM tool, probably the single most important thing you need to tell Hibernate is how to connect to your database so that it may connect on behalf of your application. This is ultimately the function of the org.hibernate.engine.jdbc.connections.spi.ConnectionProvider interface. Hibernate provides some out of the box implementations of this interface. ConnectionProvider is also an extension point so you can also use custom implementations from third parties or written yourself. The ConnectionProvider to use is defined by the hibernate.connection.provider\_class setting. See the org.hibernate.cfg.AvailableSettings#CONNECTION\_PROVIDER Generally speaking, applications should not have to configure a ConnectionProvider explicitly if using one of the Hibernate-provided implementations. Hibernate will internally determine which ConnectionProvider to use based on the following algorithm: If hibernate.connection.provider\_class is set, it takes precedence else if hibernate.connection.datasource is set → Using DataSources else if any setting prefixed by hibernate.c3p0. is set → Using c3p0 else if any setting prefixed by hibernate.proxool. is set → Using Proxool else if any setting prefixed by hibernate.hikari. is set → Using HikariCP else if any setting prefixed by hibernate.vibur. is set → Using Vibur DBCP else if any setting prefixed by hibernate.agroal. is set → Using Agroal else if hibernate.connection.url is set → Using Hibernate’s built-in (and unsupported) pooling else → User-provided Connections 7.2. Using DataSources Hibernate can integrate with a javax.sql.DataSource for obtaining JDBC Connections. Applications would tell Hibernate about the DataSource via the (required) hibernate.connection.datasource setting which can either specify a JNDI name or would reference the actual DataSource instance. For cases where a JNDI name is given, be sure to read JNDI. For Jakarta Persistence applications, note that hibernate.connection.datasource corresponds to jakarta.persistence.jtaDataSource or jakarta.persistence.nonJtaDataSource. The DataSource ConnectionProvider also (optionally) accepts the hibernate.connection.username and hibernate.connection.password. If specified, the DataSource#getConnection(String username, String password) will be used. Otherwise, the no-arg form is used. 7.3. Driver Configuration hibernate.connection.driver\_class The name of the JDBC Driver class to use hibernate.connection.url The JDBC connection url hibernate.connection.\* All such setting names (except the predefined ones) will have the hibernate.connection. prefix stripped. The remaining name and the original value will be passed to the driver as a JDBC connection property Not all properties apply to all situations. For example, if you are providing a data source, hibernate.connection.driver\_class setting will not be used. 7.4. Using c3p0 To use the c3p0 integration, the application must include the hibernate-c3p0 module jar (as well as its dependencies) on the classpath. Hibernate also provides support for applications to use c3p0 connection pooling. When c3p0 support is enabled, a number of c3p0-specific configuration settings are recognized in addition to the general ones described in Driver Configuration. Transaction isolation of the Connections is managed by the ConnectionProvider itself. See ConnectionProvider support for transaction isolation setting. hibernate.c3p0.min\_size or c3p0.minPoolSize The minimum size of the c3p0 pool. See c3p0 minPoolSize hibernate.c3p0.max\_size or c3p0.maxPoolSize The maximum size of the c3p0 pool. See c3p0 maxPoolSize hibernate.c3p0.timeout or c3p0.maxIdleTime The Connection idle time. See c3p0 maxIdleTime hibernate.c3p0.max\_statements or c3p0.maxStatements Controls the c3p0 PreparedStatement cache size (if using). See c3p0 maxStatements hibernate.c3p0.acquire\_increment or c3p0.acquireIncrement Number of connections c3p0 should acquire at a time when the pool is exhausted. See c3p0 acquireIncrement hibernate.c3p0.idle\_test\_period or c3p0.idleConnectionTestPeriod Idle time before a c3p0 pooled connection is validated. See c3p0 idleConnectionTestPeriod hibernate.c3p0.initialPoolSize The initial c3p0 pool size. If not specified, default is to use the min pool size. See c3p0 initialPoolSize Any other settings prefixed with hibernate.c3p0. Will have the hibernate. portion stripped and be passed to c3p0. Any other settings prefixed with c3p0. Get passed to c3p0 as is. See c3p0 configuration 7.5. Using Proxool To use the Proxool integration, the application must include the hibernate-proxool module jar (as well as its dependencies) on the classpath. Hibernate also provides support for applications to use Proxool connection pooling. Transaction isolation of the Connections is managed by the ConnectionProvider itself. See ConnectionProvider support for transaction isolation setting. 7.5.1. Using existing Proxool pools Controlled by the hibernate.proxool.existing\_pool setting. If set to true, this ConnectionProvider will use an already existing Proxool pool by alias as indicated by the hibernate.proxool.pool\_alias setting. 7.5.2. Configuring Proxool via XML The hibernate.proxool.xml setting names a Proxool configuration XML file to be loaded as a classpath resource and loaded by Proxool’s JAXPConfigurator. See proxool configuration. hibernate.proxool.pool\_alias must be set to indicate which pool to use. 7.5.3. Configuring Proxool via Properties The hibernate.proxool.properties setting names a Proxool configuration properties file to be loaded as a classpath resource and loaded by Proxool’s PropertyConfigurator. See proxool configuration. hibernate.proxool.pool\_alias must be set to indicate which pool to use. 7.6. Using HikariCP To use the HikariCP this integration, the application must include the hibernate-hikari module jar (as well as its dependencies) on the classpath. Hibernate also provides support for applications to use HikariCP connection pool. Set all of your Hikari settings in Hibernate prefixed by hibernate.hikari. and this ConnectionProvider will pick them up and pass them along to Hikari. Additionally, this ConnectionProvider will pick up the following Hibernate-specific properties and map them to the corresponding Hikari ones (any hibernate.hikari. prefixed ones have precedence): hibernate.connection.driver\_class Mapped to Hikari’s driverClassName setting hibernate.connection.url Mapped to Hikari’s jdbcUrl setting hibernate.connection.username Mapped to Hikari’s username setting hibernate.connection.password Mapped to Hikari’s password setting hibernate.connection.isolation Mapped to Hikari’s transactionIsolation setting. See ConnectionProvider support for transaction isolation setting. Note that Hikari only supports JDBC standard isolation levels (apparently). hibernate.connection.autocommit Mapped to Hikari’s autoCommit setting 7.7. Using Vibur DBCP To use the Vibur DBCP integration, the application must include the hibernate-vibur module jar (as well as its dependencies) on the classpath. Hibernate also provides support for applications to use Vibur DBCP connection pool. Set all of your Vibur settings in Hibernate prefixed by hibernate.vibur. and this ConnectionProvider will pick them up and pass them along to Vibur DBCP. Additionally, this ConnectionProvider will pick up the following Hibernate-specific properties and map them to the corresponding Vibur ones (any hibernate.vibur. prefixed ones have precedence): hibernate.connection.driver\_class Mapped to Vibur’s driverClassName setting hibernate.connection.url Mapped to Vibur’s jdbcUrl setting hibernate.connection.username Mapped to Vibur’s username setting hibernate.connection.password Mapped to Vibur’s password setting hibernate.connection.isolation Mapped to Vibur’s defaultTransactionIsolationValue setting. See ConnectionProvider support for transaction isolation setting. hibernate.connection.autocommit Mapped to Vibur’s defaultAutoCommit setting 7.8. Using Agroal To use the Agroal integration, the application must include the hibernate-agroal module jar (as well as its dependencies) on the classpath. Hibernate also provides support for applications to use Agroal connection pool. Set all of your Agroal settings in Hibernate prefixed by hibernate.agroal. and this ConnectionProvider will pick them up and pass them along to Agroal connection pool. Additionally, this ConnectionProvider will pick up the following Hibernate-specific properties and map them to the corresponding Agroal ones (any hibernate.agroal. prefixed ones have precedence): hibernate.connection.driver\_class Mapped to Agroal’s driverClassName setting hibernate.connection.url Mapped to Agroal’s jdbcUrl setting hibernate.connection.username Mapped to Agroal’s principal setting hibernate.connection.password Mapped to Agroal’s credential setting hibernate.connection.isolation Mapped to Agroal’s jdbcTransactionIsolation setting. See ConnectionProvider support for transaction isolation setting. hibernate.connection.autocommit Mapped to Agroal’s autoCommit setting 7.9. Using Hibernate’s built-in (and unsupported) pooling The built-in connection pool is not supported for use in a production system. This section is here just for completeness. 7.10. User-provided Connections It is possible to use Hibernate by simply passing a Connection to use to the Session when the Session is opened. This usage is discouraged and not discussed here. 7.11. ConnectionProvider support for transaction isolation setting All of the provided ConnectionProvider implementations, other than DataSourceConnectionProvider, support consistent setting of transaction isolation for all Connections obtained from the underlying pool. The value for hibernate.connection.isolation can be specified in one of 3 formats: the integer value accepted at the JDBC level. the name of the java.sql.Connection constant field representing the isolation you would like to use. For example, TRANSACTION\_REPEATABLE\_READ for java.sql.Connection#TRANSACTION\_REPEATABLE\_READ. Not that this is only supported for JDBC standard isolation levels, not for isolation levels specific to a particular JDBC driver. a short-name version of the java.sql.Connection constant field without the TRANSACTION\_ prefix. For example, REPEATABLE\_READ for java.sql.Connection#TRANSACTION\_REPEATABLE\_READ. Again, this is only supported for JDBC standard isolation levels, not for isolation levels specific to a particular JDBC driver. 7.12. Connection handling The connection handling mode is defined by the PhysicalConnectionHandlingMode enumeration which provides the following strategies: IMMEDIATE\_ACQUISITION\_AND\_HOLD The Connection will be acquired as soon as the Session is opened and held until the Session is closed. DELAYED\_ACQUISITION\_AND\_HOLD The Connection will be acquired as soon as it is needed and then held until the Session is closed. DELAYED\_ACQUISITION\_AND\_RELEASE\_AFTER\_STATEMENT The Connection will be acquired as soon as it is needed and will be released after each statement is executed. DELAYED\_ACQUISITION\_AND\_RELEASE\_AFTER\_TRANSACTION The Connection will be acquired as soon as it is needed and will be released after each transaction is completed. If you don’t want to use the default connection handling mode, you can specify a connection handling mode via the hibernate.connection.handling\_mode configuration property. For more details, check out the Database connection properties section. 7.12.1. Transaction type and connection handling By default, the connection handling mode is given by the underlying transaction coordinator. There are two types of transactions: RESOURCE\_LOCAL (which involves a single database Connection and the transaction is controlled via the commit and rollback Connection methods) and JTA (which may involve multiple resources including database connections, JMS queues, etc). RESOURCE\_LOCAL transaction connection handling For RESOURCE\_LOCAL transactions, the connection handling mode is DELAYED\_ACQUISITION\_AND\_RELEASE\_AFTER\_TRANSACTION meaning that the database connection is acquired when needed and released after the current running transaction is either committed or rolled back. However, because Hibernate needs to make sure that the default autocommit mode is disabled on the JDBC Connection when starting a new transaction, the Connection is acquired and the autocommit mode is set to false. If you are using a connection pool DataSource that already disabled the autocommit mode for every pooled Connection, you should set the hibernate.connection.provider\_disables\_autocommit to true and the database connection acquisition will be, indeed, delayed until Hibernate needs to execute the first SQL statement. JTA transaction connection handling For JTA transactions, the connection handling mode is DELAYED\_ACQUISITION\_AND\_RELEASE\_AFTER\_STATEMENT meaning that the database connection is acquired when needed and released after each statement execution. The reason for releasing the database connection after statement execution is because some Java EE application servers report a connection leak when a method call goes from one EJB to another. However, even if the JDBC Connection is released to the pool, the Connection is still allocated to the current executing Thread, hence when executing a subsequent statement in the current running transaction, the same Connection object reference will be obtained from the pool. If the Java EE application server or JTA transaction manager supports switching from one EJB to another while the transaction gets propagated from the outer EJB to the inner one, and no connection leak false positive is being reported, then you should consider switching to DELAYED\_ACQUISITION\_AND\_RELEASE\_AFTER\_TRANSACTION via the hibernate.connection.handling\_mode configuration property. 7.12.2. User-provided connections If the current Session was created using the SessionBuilder and a JDBC Connection was provided via the SessionBuilder#connection method, then the user-provided Connection is going to be used, and the connection handling mode will be IMMEDIATE\_ACQUISITION\_AND\_HOLD. Therefore for user-provided connection, the connection is acquired right away and held until the current Session is closed, without being influenced by the Jakarta Persistence or Hibernate transaction context. 7.13. Database Dialect Although SQL is relatively standardized, each database vendor uses a subset and superset of ANSI SQL defined syntax. This is referred to as the database’s dialect. Hibernate handles variations across these dialects through its org.hibernate.dialect.Dialect class and the various subclasses for each database vendor. In most cases, Hibernate will be able to determine the proper Dialect to use by asking some questions of the JDBC Connection during bootstrap. For information on Hibernate’s ability to determine the proper Dialect to use (and your ability to influence that resolution), see Dialect resolution. If for some reason it is not able to determine the proper one or you want to use a custom Dialect, you will need to set the hibernate.dialect setting. Table 1. Provided Dialects Dialect (short name) Remarks Cache71 Support for the Caché database, version 2007.1. CockroachDB192 Support for the CockroachDB database version 19.2. CockroachDB201 Support for the CockroachDB database version 20.1. CUBRID Support for the CUBRID database, version 8.3. May work with later versions. DB2 Support for the DB2 database, version 8.2. DB297 Support for the DB2 database, version 9.7. DB2390 Support for DB2 Universal Database for OS/390, also known as DB2/390. DB2400 Support for DB2 Universal Database for iSeries, also known as DB2/400. DB2400V7R3 Support for DB2 Universal Database for i, also known as DB2/400, version 7.3 DerbyTenFive Support for the Derby database, version 10.5 DerbyTenSix Support for the Derby database, version 10.6 DerbyTenSeven Support for the Derby database, version 10.7 Firebird Support for the Firebird database FrontBase Support for the Frontbase database H2 Support for the H2 database HANACloudColumnStore Support for the SAP HANA Cloud database column store. HANAColumnStore Support for the SAP HANA database column store, version 2.x. This is the recommended dialect for the SAP HANA database. May work with SAP HANA, version 1.x HANARowStore Support for the SAP HANA database row store, version 2.x. May work with SAP HANA, version 1.x HSQL Support for the HSQL (HyperSQL) database Informix Support for the Informix database Ingres Support for the Ingres database, version 9.2 Ingres9 Support for the Ingres database, version 9.3. May work with newer versions Ingres10 Support for the Ingres database, version 10. May work with newer versions Interbase Support for the Interbase database. JDataStore Support for the JDataStore database McKoi Support for the McKoi database Mimer Support for the Mimer database, version 9.2.1. May work with newer versions MySQL5 Support for the MySQL database, version 5.x MySQL5InnoDB Support for the MySQL database, version 5.x preferring the InnoDB storage engine when exporting tables. MySQL57InnoDB Support for the MySQL database, version 5.7 preferring the InnoDB storage engine when exporting tables. May work with newer versions MariaDB Support for the MariaDB database. May work with newer versions MariaDB53 Support for the MariaDB database, version 5.3 and newer. Oracle8i Support for the Oracle database, version 8i Oracle9i Support for the Oracle database, version 9i Oracle10g Support for the Oracle database, version 10g Pointbase Support for the Pointbase database PostgresPlus Support for the Postgres Plus database PostgreSQL81 Support for the PostgrSQL database, version 8.1 PostgreSQL82 Support for the PostgreSQL database, version 8.2 PostgreSQL9 Support for the PostgreSQL database, version 9. May work with later versions. Progress Support for the Progress database, version 9.1C. May work with newer versions. SAPDB Support for the SAPDB/MAXDB database. SQLServer Support for the SQL Server 2000 database SQLServer2005 Support for the SQL Server 2005 database SQLServer2008 Support for the SQL Server 2008 database Sybase11 Support for the Sybase database, up to version 11.9.2 SybaseAnywhere Support for the Sybase Anywhere database SybaseASE15 Support for the Sybase Adaptive Server Enterprise database, version 15 SybaseASE157 Support for the Sybase Adaptive Server Enterprise database, version 15.7. May work with newer versions. Teradata Support for the Teradata database TimesTen Support for the TimesTen database, version 5.1. May work with newer versions