HIBERNATE 5.X NEW FEATURES

1) Java 8 support

Well, ok. Not all of Java 8. Specifically we have added support for Java 8 Date and Time API in regards to easily mapping attributes in your domain model using the Java 8 Date and Time API types to the database. This support is available under the dedicated hibernate-java8 artifact (to isolate Java 8 dependencies

Hibernate type (org.hibernate.type package)	JDBC type	Java type	BasicTypeRegistry key(s)
DurationType	BIGINT	java.time.Duration	Duration, java.time.Duration
InstantType	TIMESTAMP	java.time.Instant	Instant, java.time.Instant
LocalDateTimeType	TIMESTAMP	java.time.LocalDateTime	LocalDateTime, java.time.LocalDateTime
LocalDateType	DATE	java.time.LocalDate	LocalDate, java.time.LocalDate
LocalTimeType	TIME	java.time.LocalTime	LocalTime, java.time.LocalTime
OffsetDateTimeType	TIMESTAMP	java.time.OffsetDateTime	OffsetDateTime, java.time.OffsetDateTime
OffsetTimeType	TIME	java.time.OffsetTime	OffsetTime, java.time.OffsetTime
OffsetTimeType	TIMESTAMP	java.time.ZonedDateTime	ZonedDateTime, java.time.ZonedDateTime

To use these hibernate-java8 types just add the hibernate-java8 dependency to your classpath and Hibernate will take care of the rest.

Mapping Java 8 Date/Time Values

Java 8 came with a new Date/Time API, offering support for instant dates, intervals, local and zoned Date/Time immutable instances, bundled in the java.time package.

- DATE : `java.time.LocalDate`
- TIME : `java.time.LocalTime`, `java.time.OffsetTime`
- TIMESTAMP : `java.time.Instant`, `java.time.LocalDateTime`, `java.time.OffsetDateTime` and `java.time.ZonedDateTime`

Hibernate added support for the new Date/Time API in a new module, which must be included with the following Maven dependency:

Maven Dependancy:-(hibernate-java8 jar)

```
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-java8</artifactId>
   <version>${hibernate.version}</version>
</dependency>
```

The mapping between the standard SQL Date/Time types and the supported Java 8 Date/Time class types looks as follows;

DATE

java.time.LocalDate

TIME

java.time.LocalTime, java.time.OffsetTime

TIMESTAMP

java.time.Instant, java.time.LocalDateTime, java.time.OffsetDateTime and java.time.ZonedDateTime

Because the mapping between Java 8 Date/Time classes and the SQL types is implicit, there is no need to specify the @Temporal annotation. Setting it on the java.time classes throws the following exception org.hibernate.AnnotationException: @Temporal should only be set on a java.util.Date or java.util.Calendar property

2) Short-Naming

In an effort to insulate applications from refactoring efforts, Hibernate has begun to recognize "short name" values for certain configuration settings.

The following short-names are recognized:

jdbc :: (the default) says to use JDBC-based transactions (org.hibernate.resource.transaction.backend.jdbc.internal.JdbcResourceLocalTransactionCoordinatorImpl)

jta :: says to use JTA-based transactions (org.hibernate.resource.transaction.backend.jta.internal.JtaTransactionCoordinatorImpl)

3) Type handling

- Built-in org.hibernate.type.descriptor.sql.SqlTypeDescriptor implementations no longer auto-register themselves with org.hibernate.type.descriptor.sql.SqlTypeDescriptorRegistry. Applications using custom SqlTypeDescriptor implementations extending the built-in ones and relying on that behavior should be updated to call SqlTypeDescriptorRegistry#addDescriptor themselves.
- The JDBC type for "big_integer" (org.hibernate.type.BigIntegerType) properties has changed from java.sql.Types.NUMERIC to java.sql.Types.BIGINT. This change was reverted in 5.0.1.Final.
- For ids defined as UUID with generation, for some databases it is required to explicitly set the @Column(length=16) in order to generate BINARY(16) so that comparisons properly work.

- For EnumType mappings defined in hbm.xml where the user wants name-mapping (javax.persistence.EnumType#STRING) the configuration must explicitly state that using either the useNamed (true) setting or by specifying the type setting set to the value 12 (VARCHAR JDBC type code).

4) Naming strategies

Historically Hibernate provided just a singular contract for applying a "naming strategy". Starting in 5.0 this has been split into 2 distinct contracts: * ImplicitNamingStrategy - is used whenever a table or column is not explicitly named to determine the name to use. * PhysicalNamingStrategy - is used to convert a "logical name" (either implicit or explicit) name of a table or column into a physical name

5) Changed setting defaults

- Default value for hibernate.id.new_generator_mappings setting changed to true for 5.0. See
  org.hibernate.cfg.AvailableSettings#USE_NEW_ID_GENERATOR_MAPPINGS javadocs.
  //from javaDoc
  static final String USE_NEW_ID_GENERATOR_MAPPINGS
  Setting which indicates whether or not the new IdentifierGenerator are used for AUTO, TABLE and SEQUENCE.Default is true. Existing applications may want to disable this (set it false) for upgrade compatibility.

- The default ImplicitNamingStrategy (hibernate.implicit_naming_strategy) has changed to the JPA-compliant one. See
  org.hibernate.cfg.AvailableSettings.IMPLICIT_NAMING_STRATEGY javadocs for details.
  `static final String IMPLICIT_NAMING_STRATEGY`
  Used to specify the `ImplicitNamingStrategy` class to use. The following short-names are defined for this setting:

  ✓ "default" -> `ImplicitNamingStrategyJpaCompliantImpl`
  ✓ "jpa" -> `ImplicitNamingStrategyJpaCompliantImpl`
  ✓ "legacy-jpa" -> `ImplicitNamingStrategyLegacyJpaImpl`

- ✓ "legacy-hbm" -> `ImplicitNamingStrategyLegacyHbmImpl`
- ✓ "component-path" -> `ImplicitNamingStrategyComponentPathImpl`

  The default is defined by the ImplicitNamingStrategy registered under the "default" key. If that happens to be empty, the fallback is to use `ImplicitNamingStrategyJpaCompliantImpl`.

- hibernate.jdbc.batch_versioned_data default value is now true . Oracle dialects set this property to false, except for Oracle12cDialect

6) Misc

- **cfg.xml** files are again fully parsed and integrated (events, security, etc)
- Properties loaded from cfg.xml through EntityManagerFactory did not previously prefix names with "**hibernate."** this is now made consistent.
- Configuration is no longer Serializable
- **org.hibernate.dialect.Dialect.getQuerySequencesString** expected to retrieve catalog, schema, and increment values as well
- Moving **org.hibernate.hql.spi.MultiTableBulkIdStrategy** and friends to new org.hibernate.hql.spi.id package and sub-packages
- Complete redesign of "property access" contracts
- Valid **hibernate.cache.default_cache_concurrency_strategy** setting values are now defined via **org.hibernate.cache.spi.access.AccessType#getExternalName** rather than the **org.hibernate.cache.spi.access.AccessType enum names**; this is more consistent with other Hibernate settings

7) Deprecations

- Removed the deprecated `org.hibernate.cfg.AnnotationConfiguration`
- Removed deprecated `org.hibernate.id.TableGenerator` id-generator

- Removed deprecated `org.hibernate.id.TableHiLoGenerator` (hilo) id-generator
- Deprecated `org.hibernate.id.SequenceGenerator` and its subclasses
- Added a new dedicated "deprecation logger" to consolidate logging for deprecated uses.

8) Changed/Moved contracts

- org.hibernate.integrator.spi.Integrator contract changed to account for bootstrap redesign
- Extracted org.hibernate.engine.jdbc.env.spi.JdbcEnvironment from JdbcServices; created org.hibernate.engine.jdbc.env package and moved a few contracts there.
- Introduction of org.hibernate.boot.model.relational.ExportableProducer which will effect anyorg.hibernate.id.PersistentIdentifierGenerator implementations
- Changed to signature of org.hibernate.id.Configurable to accept ServiceRegistry rather than just Dialect

9) Oracle12cDialect maps byte[] and Byte[] to BLOB

Previous versions of Hibernate have mapped byte[] and Byte[] to Oracle's LONG RAW data type (via the JDBC LONGVARBINARY type). Oracle has deprecated the LONG RAW data type for many releases - possibly as far back as 8i. Therefore it was decided to start having Hibernate map byte[] and Byte[] to BLOB for Oracle.
However, in the interest of backwards compatibility and not breaking existing applications it was also decided to limit this change to just the Oracle12cDialect. So starting in 5.1.0.Final applications using Oracle12cDialect and implicitly mapping byte[] and Byte[] values will start seeing those handled as BLOB data rather than LONG RAW data. For existing applications that want to continue to use Oracle12cDialect and still continue to implicitly map byte[] and Byte[] attributes to LONG RAW, there is a new configuration setting you can use to enable that: hibernate.dialect.oracle.prefer_longvarbinary, which is false by default (map to BLOB).

10) Changes To Schema Management Tooling

The changes mainly focused on:

- Unifying handling of `hbm2ddl.auto` and Hibernate's JPA schema-generation support.
- Removing JDBC concerns from the SPI to facilitate true replacement (for OGM)

These changes will only be a migration concern for applications directly using any of the following classes:

- `org.hibernate.tool.hbm2ddl.SchemaExport`
- `org.hibernate.tool.hbm2ddl.SchemaUpdate`
- `org.hibernate.tool.hbm2ddl.SchemaValidator`
- `org.hibernate.tool.schema.spi.SchemaManagementTool` or any of its delegates

11) The bytecode enhancement mechanism was redesigned from scratch.

There are three main aspects which we can enhance with bytecode instrumentation:

Lazy initialization:: Fields can be declared as `LAZY` and they will be fetched only when being accessed for the first time. Dirty checking:: Entities are enhanced so that they can keep track of all the properties that get changed after being loaded in a Persistence Context. Bidirectional associations:: It's possible to synchronize both sides of a bidirectional association automatically, even if the developer only updates a single side.

12) Hibernate’s native APIs (Like Session) have been updated to use generic typed. No need to cast when fetching entities.

13) Second level cache by reference. This features enables direct storage of entity references into the second level cache for immutable entities.