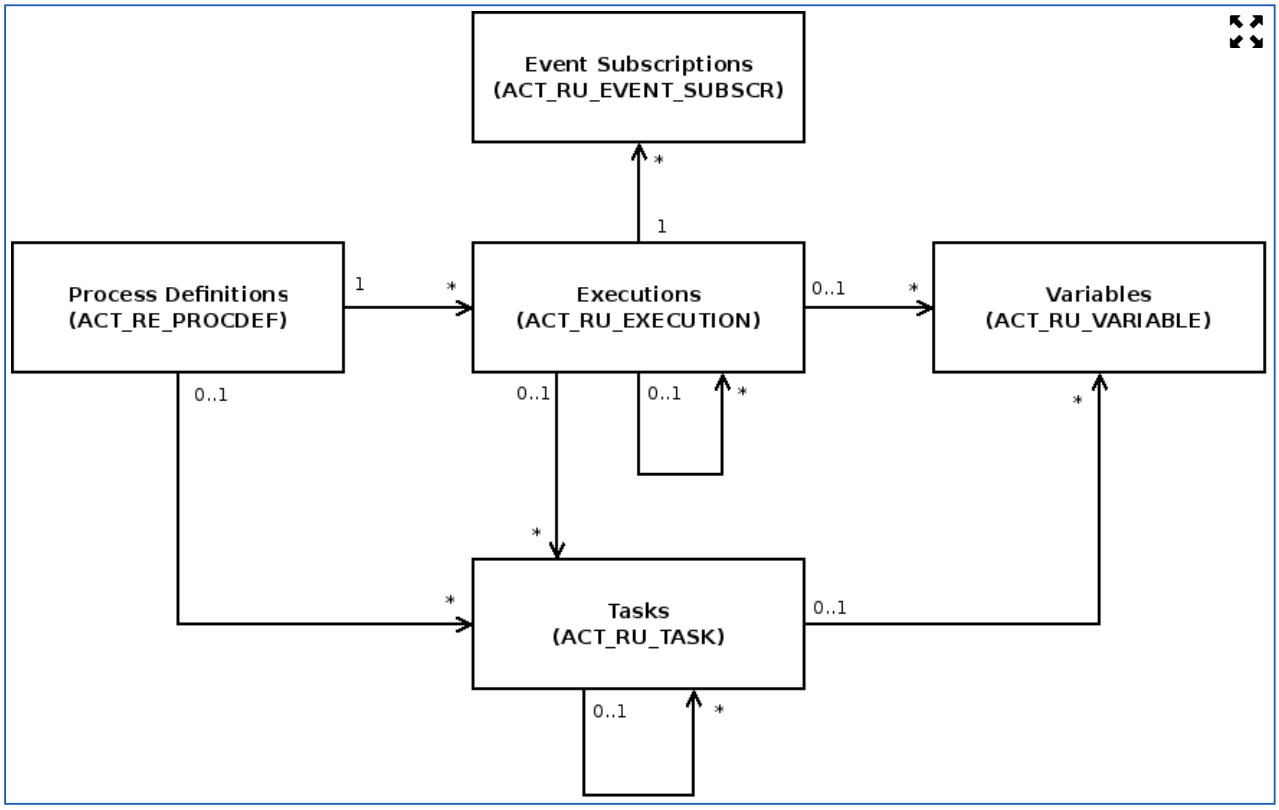
Database

Database Schema

The database schema of the process engine consists of multiple tables. The table names all start with ACT. The second part is a two-character identification of the use case of the table. This use case will also roughly match the service API.

- `ACT_RE_*`: `RE` stands for repository. Tables with this prefix contain 'static' information such as process definitions and process resources (images, rules, etc.).
- `ACT_RU_*`: `RU` stands for runtime. These are the runtime tables that contain the runtime data of process instances, user tasks, variables, jobs, etc. The engine only stores the runtime data during process instance execution and removes the records when a process instance ends. This keeps the runtime tables small and fast.
- `ACT_ID_*`: `ID` stands for identity. These tables contain identity information such as users, groups, etc.
- `ACT_HI_*`: `HI` stands for history. These are the tables that contain historical data such as past process instances, variables, tasks, etc.
- `ACT_GE_*`: General data, which is used in various use cases.

The main tables of the process engines are the entities of process definitions, executions, tasks, variables and event subscriptions. Their relationship is shown in the following UML model.



Process Definitions (ACT_RE_PROCDEF)

The ACT_RE_PROCDEF table contains all deployed process definitions. It includes information like the version details, the resource name or the suspension state.

Executions (ACT_RU_EXECUTION)

The ACT_RU_EXECUTION table contains all current executions. It includes information like the process definition, parent execution, business key, the current activity and different metadata about the state of the execution.

Tasks (ACT_RU_TASK)

The ACT_RU_TASK table contains all open tasks of all running process instances. It includes information like the corresponding process instance, execution and also metadata such as creation time, assignee or due date.

Variables (ACT_RU_VARIABLE)

The ACT_RU_VARIABLE table contains all currently set process or task variables. It includes the names, types and values of the variables and information about the corresponding process instance or task.

Event Subscriptions (ACT_RU_EVENT_SUBSCR)

The ACT_RU_EVENT_SUBSCR table contains all currently existing event subscriptions. It includes the type, name and configuration of the expected event along with information about the corresponding process instance and execution.

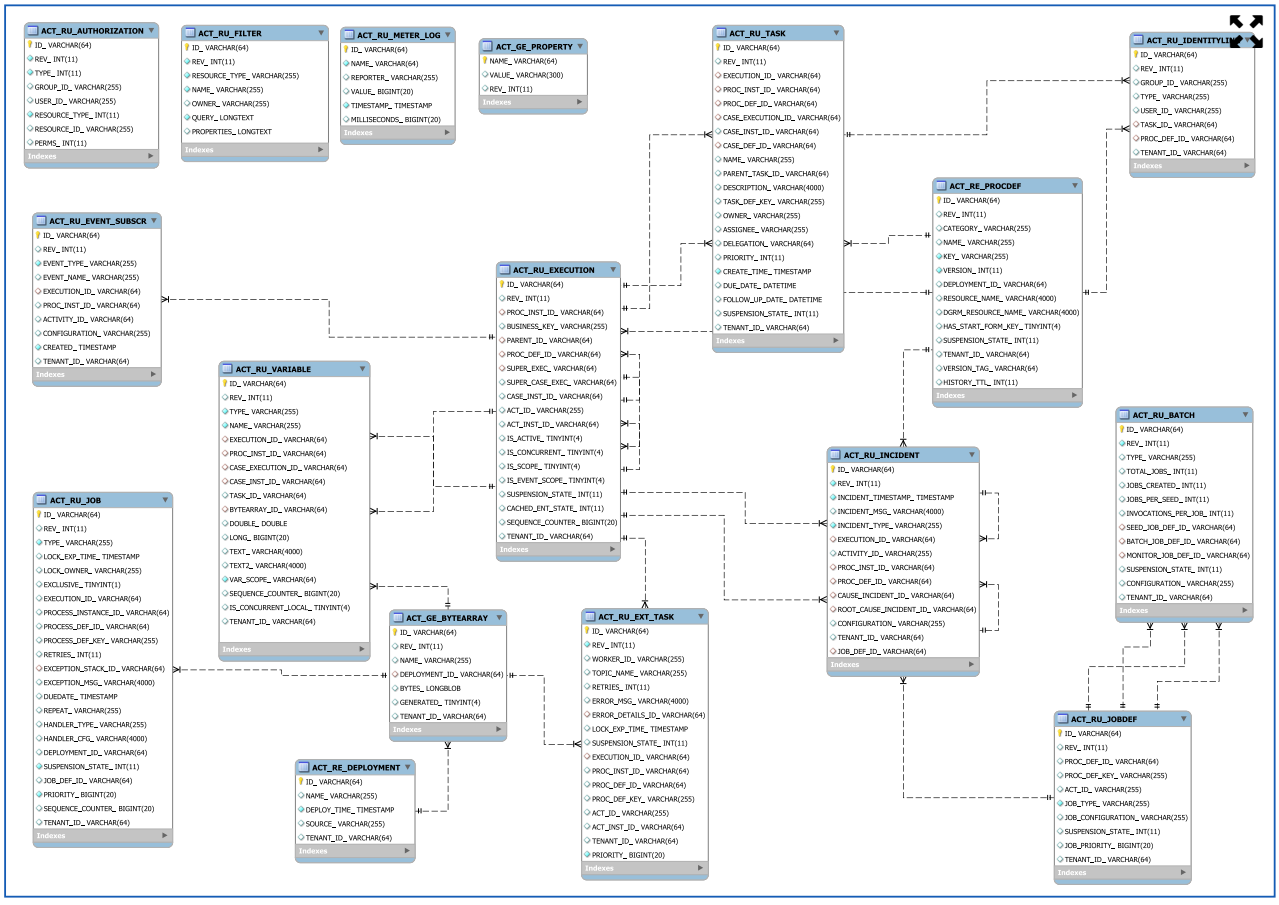
Entity Relationship Diagrams

The database is not part of the **public API**. The database schema may change for MINOR and MAJOR version updates.

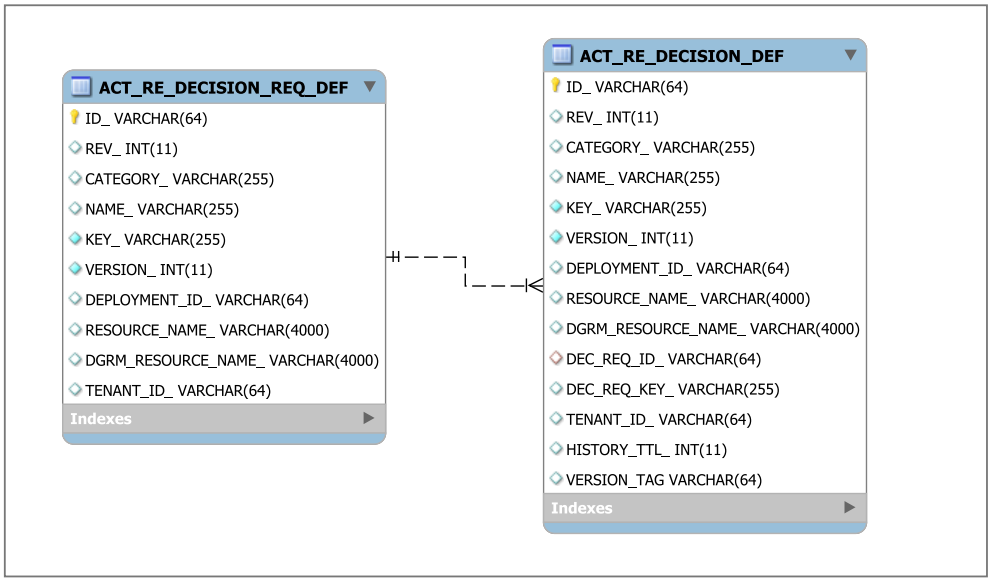
Please note: The following diagrams are based on the MySQL database schema. For other databases the diagram may be slightly different.

The following Entity Relationship Diagrams visualize the database tables and their explicit foreign key constraints, grouped by Engine with focus on BPMN, Engine with focus on DMN, Engine with focus on CMMN, the Engine History and the Identity. Please note that the diagrams do not visualize implicit connections between the tables.

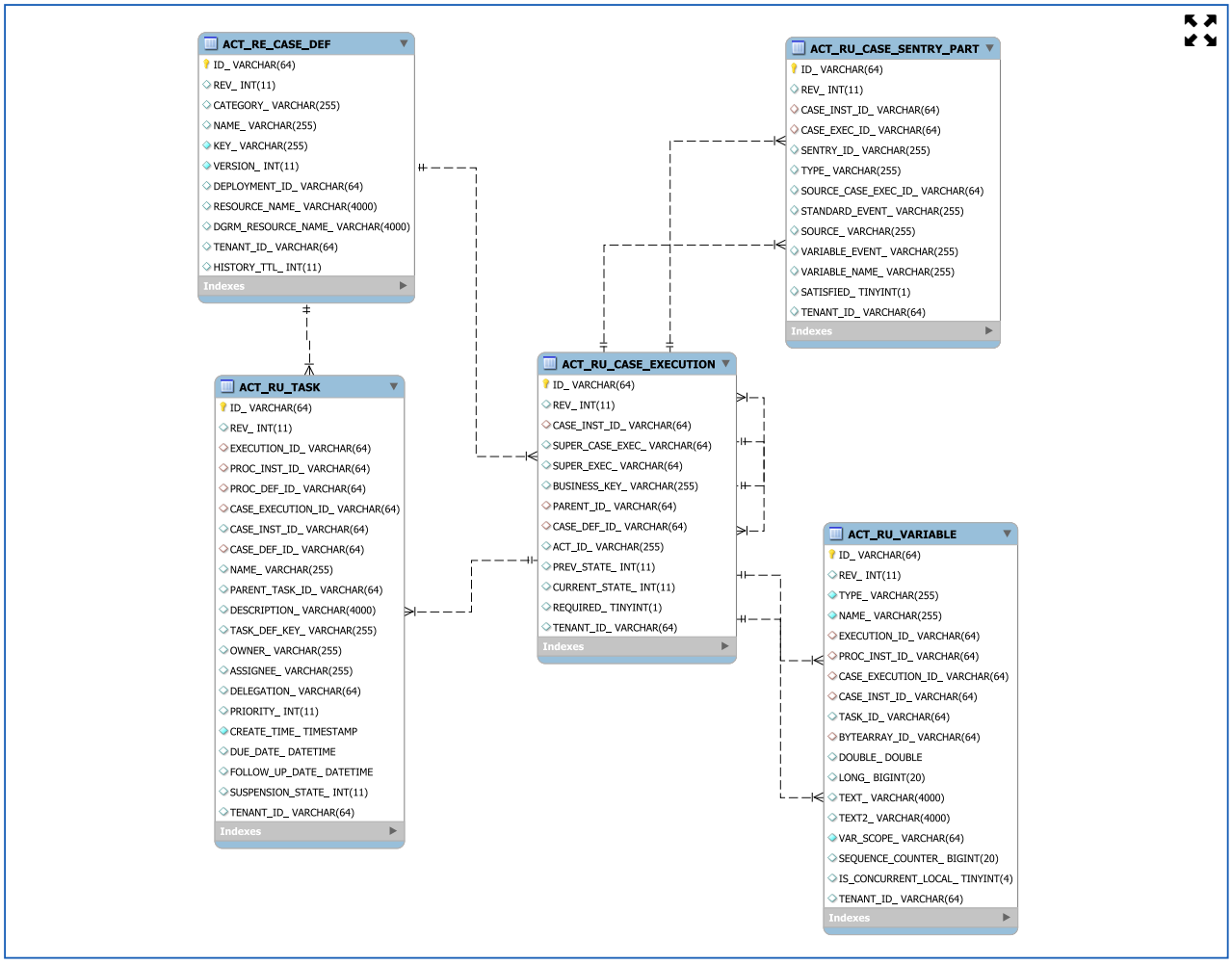
Engine BPMN



Engine DMN

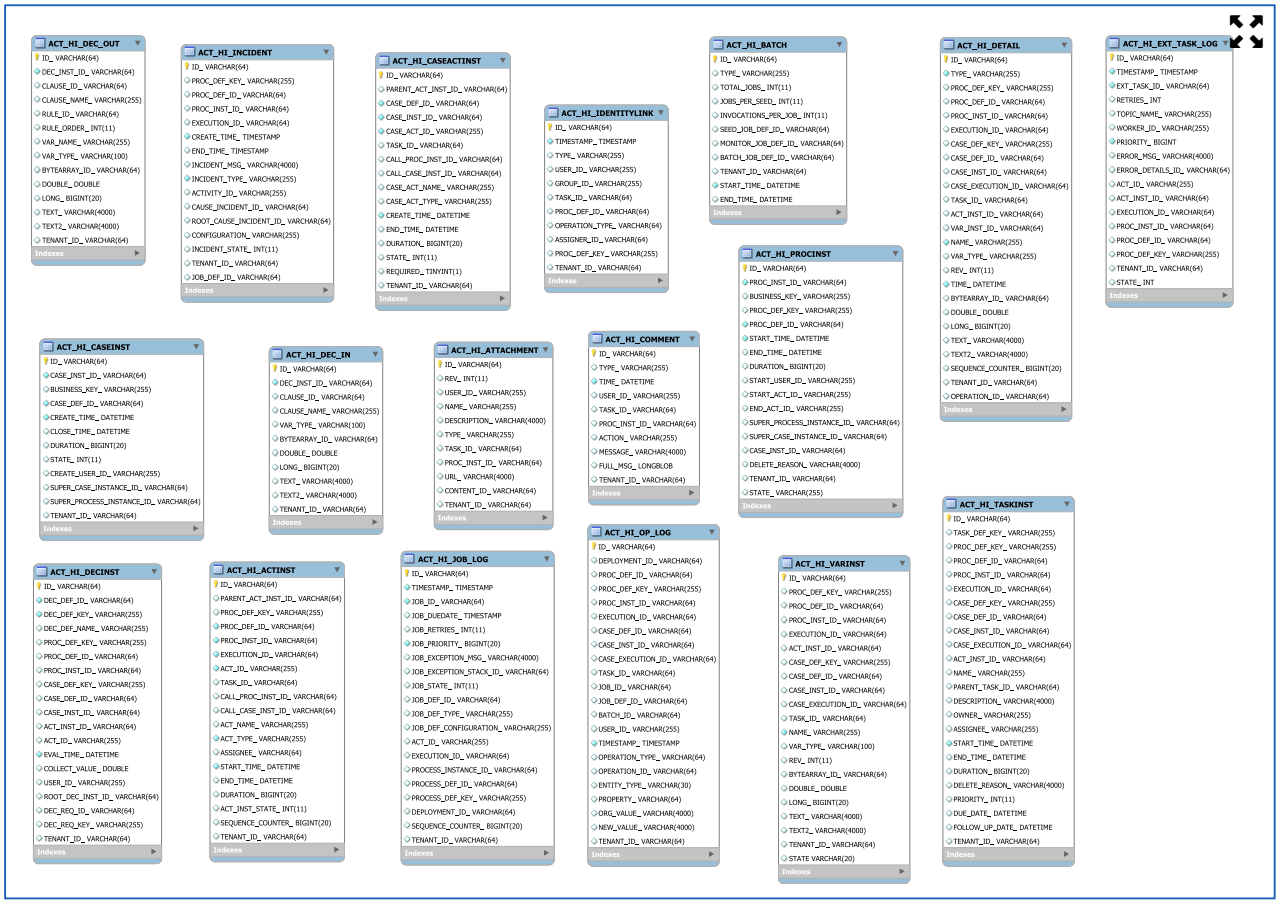


Engine CMMN

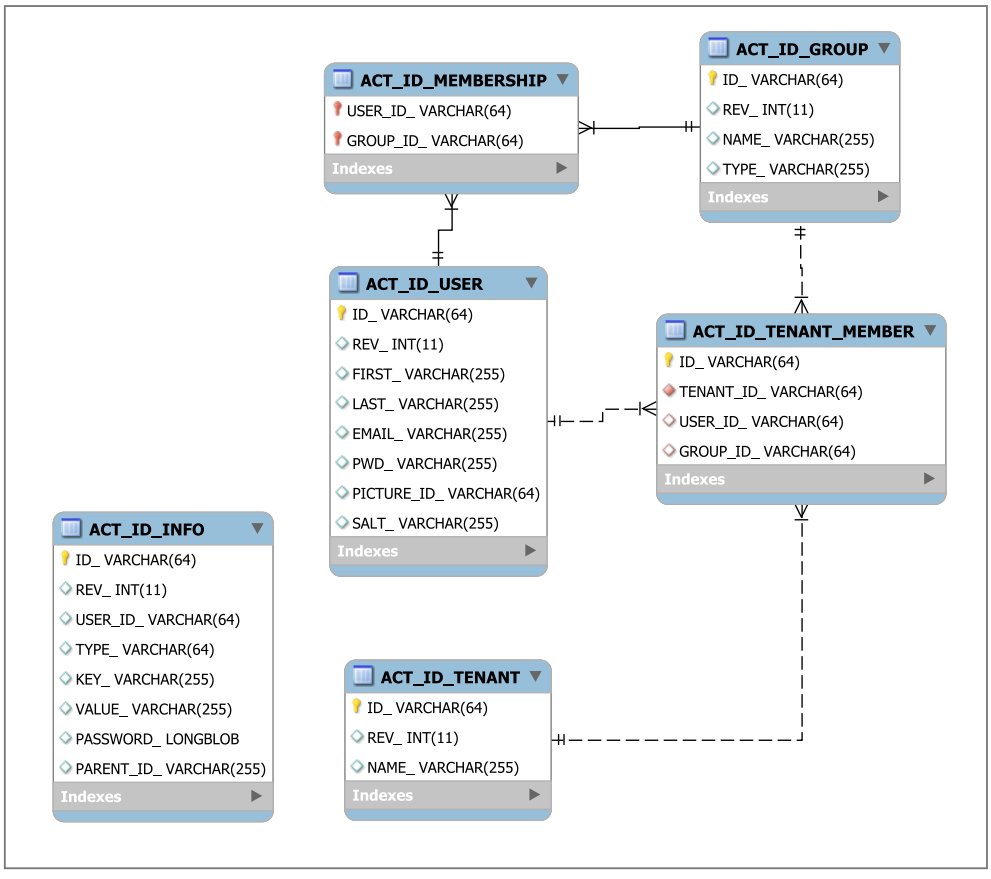


History

To allow different configurations and to keep the tables more flexible, the history tables contain no foreign key constraints.



Identity



Database Configuration

There are two ways to configure the database that the Camunda engine will use. The first option is to define the JDBC properties of the database:

- `jdbcUrl`: JDBC URL of the database.
- `jdbcDriver`: implementation of the driver for the specific database type.
- `jdbcUsername`: username to connect to the database.
- `jdbcPassword`: password to connect to the database.

Note that the engine uses Apache MyBatis (http://www.mybatis.org/) internally for persistence.

The data source that is constructed based on the provided JDBC properties will have the default MyBatis connection pool settings. The following attributes can optionally be set to tweak that connection pool (taken from the MyBatis documentation):

- `jdbcMaxActiveConnections`: The maximum number of active connections that the connection pool can contain at any given time. Default is 10.
- `jdbcMaxIdleConnections`: The maximum number of idle connections that the connection pool can contain at any given time.
- `jdbcMaxCheckoutTime`: The amount of time in milliseconds that a connection can be 'checked out' for from the connection pool before it is forcefully returned. Default is 20000 (20 seconds).
- `jdbcMaxWaitTime`: This is a low level setting that gives the pool a chance to print a log status and re-attempt the acquisition of a connection in the case that it takes unusually long (to avoid failing silently forever if the pool is misconfigured). Default is 20000 (20 seconds).
- `jdbcStatementTimeout`: The amount of time in seconds the JDBC driver will wait for a response from the database. Default is null which means that there is no timeout.

Jdbc Batch Processing

Another configuration - `jdbcBatchProcessing` - sets if batch processing mode must be used when sending SQL statements to the database. When switched off, statements are executed one by one. Values: `true` (default), `false`.

Known issues with batch processing:

- batch processing is not working for Oracle versions earlier than 12.
- when using batch processing on MariaDB and DB2, `jdbcStatementTimeout` is being ignored.

Example database configuration

```
<property name="jdbcUrl" value="jdbc:h2:mem:camunda;DB_CLOSE_DELAY=1000" />
<property name="jdbcDriver" value="org.h2.Driver" />
<property name="jdbcUsername" value="sa" />
<property name="jdbcPassword" value="" />
```

Alternatively, a `javax.sql.DataSource` implementation can be used (e.g., DBCP from Apache Commons):

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" >
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />
  <property name="url" value="jdbc:mysql://localhost:3306/camunda" />
  <property name="username" value="camunda" />
  <property name="password" value="camunda" />
  <property name="defaultAutoCommit" value="false" />
</bean>

<bean id="processEngineConfiguration" class="org.camunda.bpm.engine.impl.cfg.StandaloneProcessEngineConf

    <property name="dataSource" ref="dataSource" />
    ...
```

Note that Camunda does not ship with a library that allows to define such a data source. So you have to make sure that the libraries (e.g., from DBCP) are on your classpath.

The following properties can be set, regardless of whether you are using the JDBC or data source approach:

- `databaseType`: It's normally not necessary to specify this property as it is automatically analyzed from the database connection meta data. Should only be specified in case automatic detection fails. Possible values: {h2, mysql, oracle, postgres, mssql, db2, mariadb}. This setting will determine which create/drop scripts and queries will be used. See the 'supported databases' section for an overview of which types are supported.
- `databaseSchemaUpdate`: Allows to set the strategy to handle the database schema on process engine boot and shutdown.
  - `true` (default): Upon building the process engine, a check is performed whether the Camunda tables exist in the database. If they don't exist, they are created. It must be ensured that the version of the DB schema matches the version of the process engine library, unless performing a Rolling Update (/manual/7.8/update/rolling-update/). Updates of the database schema have to be done manually as described in the Update and Migration Guide (/manual/7.8/update/).
  - `false`: Does not perform any checks and assumes that the Camunda table exist in the database. It must be ensured that the version of the DB schema matches the version of the process engine library, unless performing a Rolling Update (/manual/7.8/update/rolling-update/). Updates of the database schema have to be done manually as described in the Update and Migration Guide (/manual/7.8/update/).
  - `create-drop`: Creates the schema when the process engine is being created and drops the schema when the process engine is being closed.

> Supported Databases
>
> For information on supported databases please refer to Supported Environments (/manual/7.8/introduction/supported-environments/#databases)

Here are some sample JDBC urls:

- H2: `jdbc:h2:tcp://localhost/camunda`
- MySQL: `jdbc:mysql://localhost:3306/camunda?autoReconnect=true`

- Oracle: `jdbc:oracle:thin:@localhost:1521:xe`
- PostgreSQL: `jdbc:postgresql://localhost:5432/camunda`
- DB2: `jdbc:db2://localhost:50000/camunda`
- MSSQL: `jdbc:sqlserver://localhost:1433/camunda`
- MariaDB: `jdbc:mariadb://localhost:3306/camunda`

Additional Database Schema Configuration

Business Key

Since the release of Camunda BPM 7.0.0-alpha9, the unique constraint for the business key is removed in the runtime and history tables and the database schema create and drop scripts. If you rely on the constraint, you can add it manually to your schema by issuing following sql statements:

DB2

```
Runtime:
alter table ACT_RU_EXECUTION add UNI_BUSINESS_KEY varchar (255) not null generated always as (case when "
alter table ACT_RU_EXECUTION add UNI_PROC_DEF_ID varchar (64) not null generated always as (case when "PR
create unique index ACT_UNIQ_RU_BUS_KEY on ACT_RU_EXECUTION(UNI_PROC_DEF_ID, UNI_BUSINESS_KEY);

History:
alter table ACT_HI_PROCINST add UNI_BUSINESS_KEY varchar (255) not null generated always as (case when "B
alter table ACT_HI_PROCINST add UNI_PROC_DEF_ID varchar (64) not null generated always as (case when "PRO
create unique index ACT_UNIQ_HI_BUS_KEY on ACT_HI_PROCINST(UNI_PROC_DEF_ID, UNI_BUSINESS_KEY);
```

H2

```
Runtime:
alter table ACT_RU_EXECUTION add constraint ACT_UNIQ_RU_BUS_KEY unique(PROC_DEF_ID_, BUSINESS_KEY_);

History:
alter table ACT_HI_PROCINST add constraint ACT_UNIQ_HI_BUS_KEY unique(PROC_DEF_ID_, BUSINESS_KEY_);
```

MSSQL

```
Runtime:
create unique index ACT_UNIQ_RU_BUS_KEY on ACT_RU_EXECUTION (PROC_DEF_ID_, BUSINESS_KEY_) where BUSINESS_

History:
create unique index ACT_UNIQ_HI_BUS_KEY on ACT_HI_PROCINST (PROC_DEF_ID_, BUSINESS_KEY_) where BUSINESS_K
```

MySQL

```
Runtime:
alter table ACT_RU_EXECUTION add constraint ACT_UNIQ_RU_BUS_KEY UNIQUE (PROC_DEF_ID_, BUSINESS_KEY_);

History:
alter table ACT_HI_PROCINST add constraint ACT_UNIQ_HI_BUS_KEY UNIQUE (PROC_DEF_ID_, BUSINESS_KEY_);
```

Oracle

```
Runtime:
create unique index ACT_UNIQ_RU_BUS_KEY on ACT_RU_EXECUTION
         (case when BUSINESS_KEY_ is null then null else PROC_DEF_ID_ end,
         case when BUSINESS_KEY_ is null then null else BUSINESS_KEY_ end);

History:
create unique index ACT_UNIQ_HI_BUS_KEY on ACT_HI_PROCINST
         (case when BUSINESS_KEY_ is null then null else PROC_DEF_ID_ end,
         case when BUSINESS_KEY_ is null then null else BUSINESS_KEY_ end);
```

PostgreSQL

```
Runtime:
alter table ACT_RU_EXECUTION add constraint ACT_UNIQ_RU_BUS_KEY UNIQUE (PROC_DEF_ID_, BUSINESS_KEY_);

History:
alter table ACT_HI_PROCINST add constraint ACT_UNIQ_HI_BUS_KEY UNIQUE (PROC_DEF_ID_, BUSINESS_KEY_);
```

Isolation Level Configuration

Most database management systems provide four different isolation levels to be set. For instance the levels defined by ANSI/USO SQL are (from low to high isolation):

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READS
- SERIALIZABLE

The required isolation level to run Camunda with is **READ COMMITTED**, which may have a different name according to your database system. Setting the level to REPEATABLE READS is known to cause deadlocks, so one needs to be careful, when changing the isolation level.

Configuration for Microsoft SQL Server

Microsoft SQL Server implements the isolation level `READ_COMMITTED` different than most databases and does not interact well with the process engine's optimistic locking (/manual/7.8/user-guide/process-engine/transactions-in-processes/#optimistic-locking) scheme. As a result you may suffer deadlocks when putting the process engine under high load.

If you experience deadlocks in your MSSQL installation, you must execute the following statements in order to enable SNAPSHOT isolation:

```
ALTER DATABASE [process-engine]
SET ALLOW_SNAPSHOT_ISOLATION ON

ALTER DATABASE [process-engine]
SET READ_COMMITTED_SNAPSHOT ON
```

where `[process-engine]` contains the name of your database.

Configuration for MariaDB Galera Cluster

This section documents the supported Galera Cluster configuration for MariaDB. Both server and client need to be configured correctly. Please note that there are some known limitations which apply when using Galera cluster, see below.

> Warning
>
> Please note that server and client configuration settings defined below are the only configuration that is supported for Galera Cluster. Other configurations are not supported.

Server Configuration

The following configuration needs to go into the `[galera]` configuration section in the `my.cnf.d/server.cnf` on each server:

```
[galera]
binlog_format=row
default_storage_engine=InnoDB
innodb_autoinc_lock_mode=2
transaction-isolation=READ-COMMITTED
wsrep_on=ON
wsrep_causal_reads = 1
wsrep_sync_wait = 7
...
```

Note that other setting may be present in this section but the settings `transaction-isolation`, `wsrep_on`, `wsrep_causal_reads` and `wsrep_sync_wait` need to present and need to have **exactly** the values shown above.

Client Configuration

Only `failover` and `sequential` configurations are supported. **Other client configuration modes like `replication:`, `loadbalance:`, `aurora:` are not supported.**

The following is the required format of the jdbcUrl property in datasource configurations:

```
jdbc:mariadb:[failover|sequential]://[host1:port],[host2:port],.../[data-base-name]
```

Examples:

```
jdbc:mariadb:failover://192.168.1.1:32980,192.168.1.2:32980,192.168.1.3:32980/process-en
gine
jdbc:mariadb:sequential://192.168.1.1:32980,192.168.1.2:32980,192.168.1.3:32980/process-
engine
```

Important: when running Camunda in a cluster, the client configuration needs to be the same on each node.

Galera Cluster Known Limitations

The following known limitations apply when using Galera Cluster:

1. APIs requiring Pessimistic read locks in the database do not work correctly. Affected APIs: Exclusive Message correlation (`.correlateExclusively()`). See (Javadocs (/manual/7.8/reference/javadoc/?org/camunda/bpm/engine/runtime/MessageCorrelationBuilder.html#correlateExclusively%28%29)). Another possible negative effects:
   - duplication of deployed definitions when deploying the same resources from two threads simultaneously
   - duplication of history cleanup job when calling `HistoryService#cleanUpHistoryAsync` from two threads simultaniousely
2. Duplicate checking during deployment does not work if resources are deployed in a cluster concurrently. Concrete impact: Concrete impact: suppose there is a Camunda process engine cluster which connects to the same Galera cluster. On deployment of a new process application the process engine nodes will check if the BPMN processes provided by the process application are already deployed, to avoid duplicate deployments. If the deployment is done simultaneously on multiple process engine nodes an exclusive read lock is acquired on the the database (technically, this means that each node performs an SQL `select for update` query.), to do the duplicate checking reliably under concurrency. This does not work on Galera Cluster and may lead to multiple versions of the same process being deployed.
3. The `jdbcStatementTimeout` configuration setting does not work and cannot be used.