**Liquibase** — это система управления миграциями базы данных.Ккросс платформенное Java приложение. Позволяет автоматизировать внесение обновлений в структуру БД. Каждое изменение описывается в декларативном стиле и версионируется.

**Преимущества**

* Универсальность. Так как changeset-ы в основном пишутся на xml и могут быть применены к разным СУБД.
* Автогенерация миграций (не всегда хорошо).
* В простейших случаях может автоматически генерировать rollback.

#### Changelog

Изменения структуры базы данных записываются в файлы, которые называются changelog.

Поддерживаемые форматы: XML, YAML, JSON или SQL.

Файлы изменений могут быть произвольно включены друг в друга для лучшего управления.

**ChangeSet** – это аналог коммита в системах контроля версий, таких как Git. **ChangeSet** может содержать одно или несколько изменений базы данных. Хорошей практикой считается одна команда для одного ChangeSet.

Каждый changeSet имеет составной идентификатор **id, author** и filename, который должен быть уникальным.

При первом запуске Liquibase создает две технические таблицы:

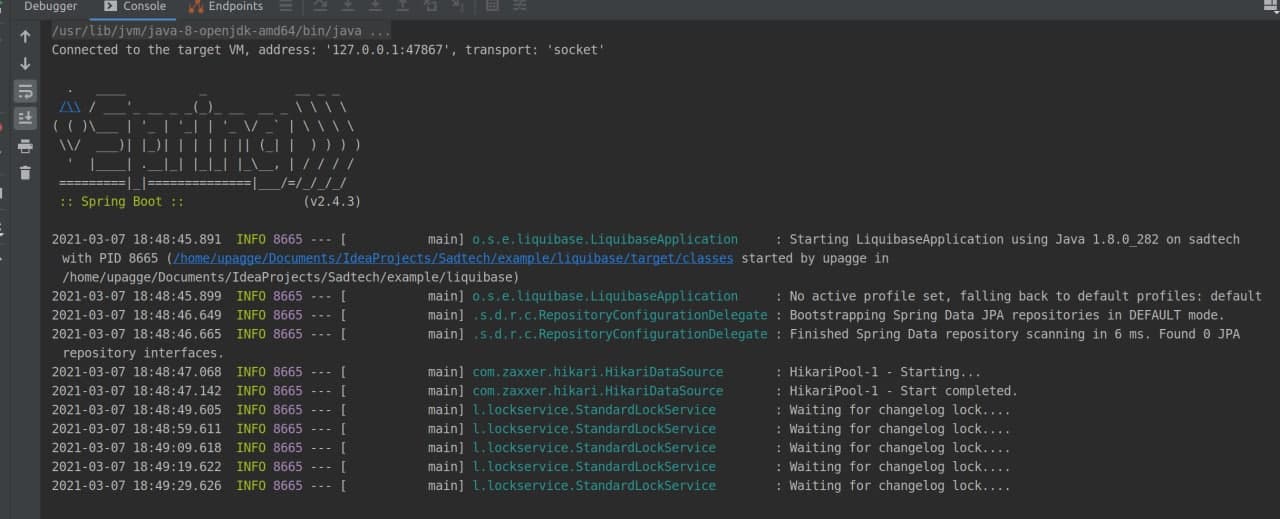
* databasechangelog – Содержит список изменений схемы БД. Туда записываются уже выполненные changeSet.
* databasechangelock – Используется для блокировки на время работы, чтобы гарантировать одновременную работу только одного экземпляра Liquibase.

**Блокировка**

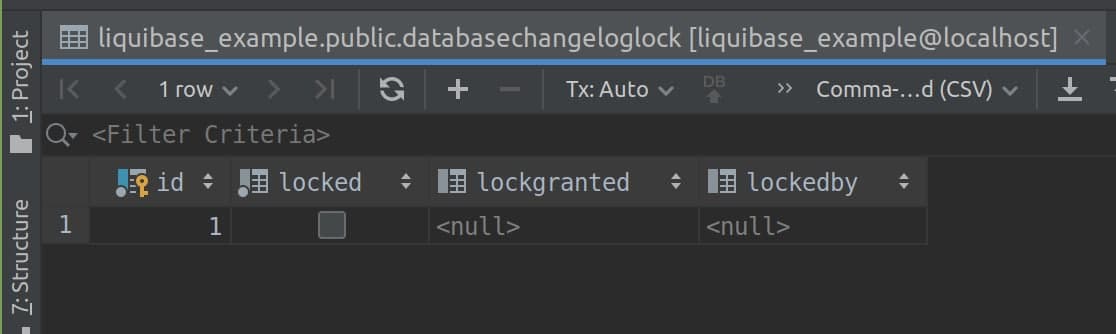
Если несколько экземпляров Liquibase будут выполняться одновременно с одной и той же базой данных, вы получите конфликты. Это может произойти, если несколько разработчиков используют один и тот же экземпляр базы данных или если в кластере несколько серверов, которые автоматически запускают Liquibase при запуске.

Для защиты от таких ситуаций Liquibase создает таблицу databasechangelock, в которой есть boolean поле locked. При запуске Liquibase проверяет его состояние, и если оно true, то ожидает смены на false.

Экстренно остановив выполнение программы в самом начале, может сложиться ситуация при котором Liquibase успеет поставить флаг, но не поменяет его на false. В логах это будет выглядеть так:



Чтобы исправить эту проблему, в таблице databasechangelock измените поле locked на false.

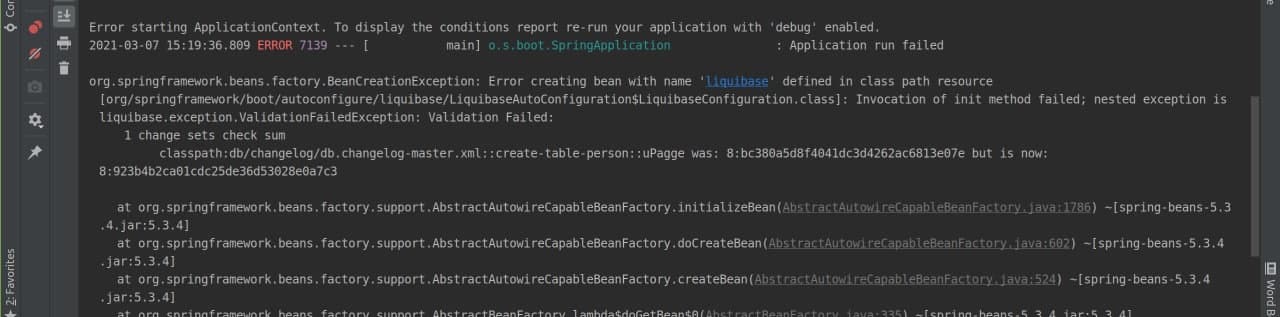


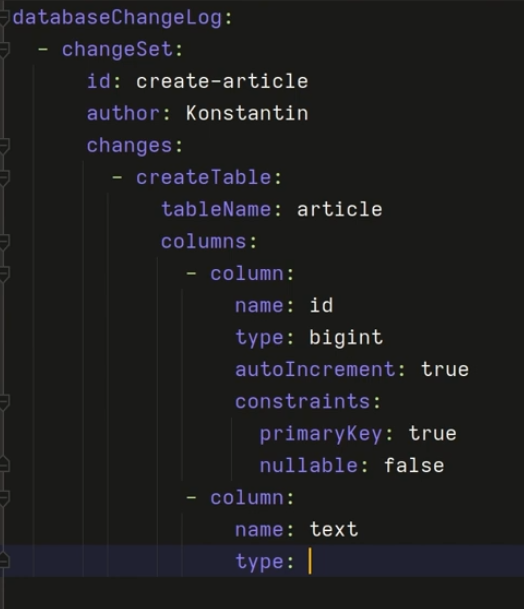
**Контрольная сумма**

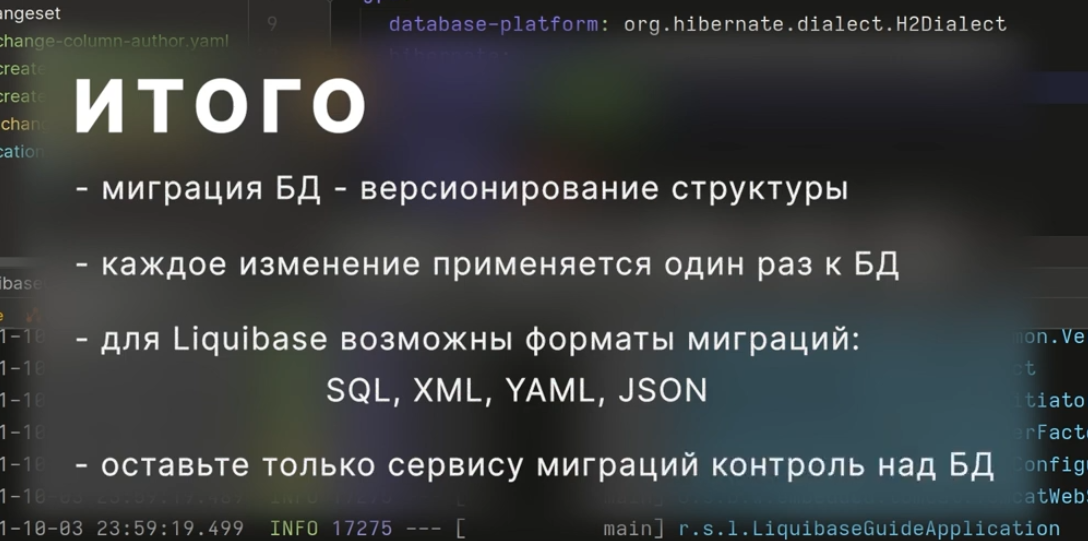
Далее Liquibase читает главный changelog, проверяя какие изменения уже были приняты, а какие надо выполнить.

После выполнения changeSet в таблицу databasechangelog со всем прочим записывается MD5 хэш changeSet. Хэш высчитывается на основе нормализованного содержимого XML.

При следующем запуске Liquibase будет сверять вновь рассчитанные хэш суммы, со значениями в его таблице. *Если вы изменили уже выполненный changeSet, то хэш сумма будет отличаться, и приложение упадет с ошибкой при старте.*





 Наиболее важным применением Liquibase является синхронизация изменений базы данных с несколькими разработчиками в команде; когда вы вносите некоторые изменения в локальную базу данных, другие разработчики могут применить эти изменения и применить их на своих машинах. Ваши изменения будут проверяться в каждом наборе изменений, и вы можете легко вернуться к предыдущему состоянию, если что-то пойдет не так. Кроме того, вы можете использовать Liquibase для миграции и развертывания баз данных.

**Liquibase умеет генерировать**

* чейнджсеты по имеющейся схеме базы данных,
* разницу, произошедшие в базе данных по сравнению с имеющимися чейнджсетами (применимо для Grails).

**КОМАНДЫ Liquibase**

* **liquibase update** – накатить чейнджсеты
* **liquibase generateChangeLog** – сгенерировать changelog из уже существующей бд
* **liquibase rollbackCount 1 –** откатиться на указанное число чейнджсетов.
* **liquibase tag checkpoint\_1 –** пометить состояние бд тегом.
* **liquibase rollback checkpoint\_1 –** откатиться к указанному тегу.

в файл application.yml укажем соединение с базой данных:

spring:

datasource:

url: jdbc:postgresql://localhost:5432/liquibase\_example

username: postgres

driver-class-name: org.postgresql.Driver

password: password

#### Создание таблицы

Создадим таблицу Person.

<changeSet id="create-table-person" author="uPagge">

<createTable tableName="person">

<column name="id" type="int" autoIncrement="true">

<constraints nullable="false" primaryKey="true"/>

</column>

<column name="name" type="varchar(64)"/>

<column name="telegram\_id" type="int">

<constraints unique="true"/>

</column>

</createTable>

</changeSet>

Тег createTable содержит параметр tableName, который указывает имя новой таблицы. Внутри этого тега мы перечислили колонки, которые нам нужны.

Для колонок обязательно необходимо указать тип. Тип указывается в формате Liquibase, после чего он приводится для конкретной реализации БД.

Отдельного внимания заслуживает колонка id. Для нее мы задали автоинкремент, а так же в constraints указали ограничения колонки:

* primaryKey="true" – колонка является первичным ключом таблицы.
* nullable="false" – значения не могут быть NULL.

При использовании primaryKey параметр nullable не обязателен. Но если вы используете H2 для тестов, то у вас могут возникнуть проблемы из-за его отсутствия.

После запуска spring-boot приложения у нас будет создано 3 таблицы, одна из которых и будет person.

#### Добавление колонки в таблицу

А теперь попробуем добавить новую колонку в таблицу в этом changeSet. Изменим его:

<changeSet id="create-table-person" author="uPagge">

<createTable tableName="person">

<column name="id" type="int" autoIncrement="true">

<constraints nullable="false" primaryKey="true"/>

</column>

<column name="name" type="varchar(64)"/>

<column name="telegram\_id" type="int">

<constraints unique="true"/>

</column>

<column name="address" type="varchar(300)"/>

</createTable>

</changeSet>

Снова запустив приложение мы получим ошибку.

Если changeSet уже выполнился, и запись об этом есть в databasechangelog, то вы не можете просто изменить changeSet. Вы же не можете в git изменить уже опубликованный коммит.

В этом случае у вас три пути:

* Создать новый changeSet с изменениями. **[Рекомендуемый]**
* Выполнить откат средствами Liquibase.
* Удалить запись о выполнении changeSet из databasechangelog. Не рекомендую этот вариант, если changeSet уже был выполнен на каком-то контуре. Этот вариант удобен для локальной разработке.

Вернем changeSet в его предыдущее состояние и создадим новый:

<changeSet id="create-table-person" author="uPagge">

<createTable tableName="person">

<column name="id" type="int" autoIncrement="true">

<constraints nullable="false" primaryKey="true"/>

</column>

<column name="name" type="varchar(64)"/>

<column name="telegram\_id" type="int">

<constraints unique="true"/>

</column>

</createTable>

</changeSet>

<changeSet id="add-new-column-address" author="uPagge">

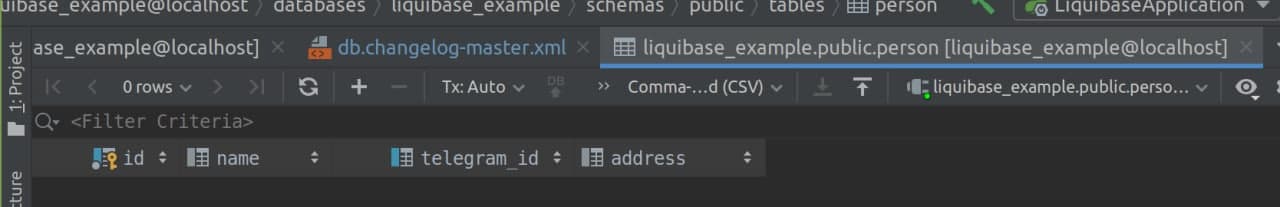
<addColumn tableName="person">

<column name="address" type="varchar(300)"/>

</addColumn>

</changeSet>

Запускаем приложение. На этот раз успешно, новая колонка добавилась.



#### Связь с другой таблицей

Связь между таблицами довольно частое явление. Добавим новую таблицу Book и свяжем ее с таблицей Person. Создадим новый changeSet:

<changeSet id="create-table-book" author="uPagge">

<createTable tableName="book">

<column name="id" type="int" autoIncrement="true">

<constraints nullable="false" primaryKey="true"/>

</column>

<column name="name" type="varchar(64)"/>

<column name="author\_id" type="int">

<constraints foreignKeyName="book\_author\_id\_person\_id" references="person(id)"/>

</column>

</createTable>

</changeSet>

Теперь атрибут author\_id связан с атрибутом id в таблице Person.

При этом обязательно нужно указать уникальный foreignKeyName. Я пользуюсь следующим правилом: имя\_таблицы + имя\_поля + имя\_главной\_таблицы + имя\_поля\_главной\_таблицы.

Также мы можем указать тип каскадной операции:

<constraints foreignKeyName="book\_author\_id\_person\_id" references="person(id)" deleteCascade="true"/>

Теперь, если автор книги будет удален, то книга тоже будет удалена.

Если вам необходима операция каскадного обновления, то вам нужен второй способ связи с таблицей:

<changeSet id="create-table-book" author="uPagge">

<createTable tableName="book">

<column name="id" type="int" autoIncrement="true">

<constraints nullable="false" primaryKey="true"/>

</column>

<column name="name" type="varchar(64)"/>

<column name="author\_id" type="int"/>

</createTable>

<addForeignKeyConstraint baseTableName="book" baseColumnNames="author\_id"

constraintName="book\_author\_id\_person\_id"

referencedTableName="person"

referencedColumnNames="id"

onUpdate="CASCADE"/>

</changeSet>

#### Связь с другой таблицей

Связь между таблицами довольно частое явление. Добавим новую таблицу Book и свяжем ее с таблицей Person. Создадим новый changeSet:

<changeSet id="create-table-book" author="uPagge">

<createTable tableName="book">

<column name="id" type="int" autoIncrement="true">

<constraints nullable="false" primaryKey="true"/>

</column>

<column name="name" type="varchar(64)"/>

<column name="author\_id" type="int">

<constraints foreignKeyName="book\_author\_id\_person\_id" references="person(id)"/>

</column>

</createTable>

</changeSet>

Теперь атрибут author\_id связан с атрибутом id в таблице Person.

При этом обязательно нужно указать уникальный foreignKeyName. Я пользуюсь следующим правилом: имя\_таблицы + имя\_поля + имя\_главной\_таблицы + имя\_поля\_главной\_таблицы.

Также мы можем указать тип каскадной операции:

<constraints foreignKeyName="book\_author\_id\_person\_id" references="person(id)" deleteCascade="true"/>

Теперь, если автор книги будет удален, то книга тоже будет удалена.

Если вам необходима операция каскадного обновления, то вам нужен второй способ связи с таблицей:

<changeSet id="create-table-book" author="uPagge">

<createTable tableName="book">

<column name="id" type="int" autoIncrement="true">

<constraints nullable="false" primaryKey="true"/>

</column>

<column name="name" type="varchar(64)"/>

<column name="author\_id" type="int"/>

</createTable>

<addForeignKeyConstraint baseTableName="book" baseColumnNames="author\_id"

constraintName="book\_author\_id\_person\_id"

referencedTableName="person"

referencedColumnNames="id"

onUpdate="CASCADE"/>

</changeSet>

## Контексты запуска

Для более удобного управления различными конфигурациями, например development/production можно использовать контексты. Контекст указывается в **changeSet** аттрибуте context и затем запускается Maven параметром -Dcontexts.

### Изменение с контекстом

<changeSet context="legacy" author="author" id="1-initial-changeset">

<tagDatabase tag="initial"/>

</changeSet>

### Запуск изменений по контексту

#!/usr/bin/env bash

mvn liquibase:update\

-Denv=dev\

-Dliquibase.url="jdbc:postgresql://dev/test?prepareThreshold=0&stringtype=unspecified"\

-Dliquibase.contexts=non-legacy

## Откат изменений

Операция обратная обновлению, в большинстве случаев поддерживается автоматически. Для прочих возможно задание через секцию rollback. Запускается командой liquibase:rollback.

### Изменение с откатом

<changeSet context="legacy" author="author" id="1-domain-some-domain">

<sql>

CREATE DOMAIN public.some\_domain AS bigint;

ALTER DOMAIN public.some\_domain OWNER TO test;

</sql>

<rollback>

DROP DOMAIN public.some\_domain;

</rollback>

</changeSet>

**RunAllways –** применяет changset всегда.

**RunOnChange –** применяет когда меняем что-то в changeset. Удобно использовать при описании хранимых процедур или представлений. Чтобы при изменении changeset не падал liquibase.



# Таблицы

# DATABASECHANGELOGLOCK

Liquibase использует таблицу DATABASECHANGELOGLOCK, чтобы обеспечить одновременное выполнение только одного экземпляра Liquibase.

Когда вы делаете обновление базы данных, Liquibase считывает данные из таблицы DATABASECHANGELOG, чтобы определить, какие наборы изменений необходимо выполнить. Чтобы избежать конфликтов между одновременными обновлениями — что может произойти, если несколько разработчиков используют один и тот же экземпляр базы данных или если несколько серверов в кластере автоматически запускают Liquibase при запуске — таблица DATABASECHANGELOGLOCK устанавливает для столбца LOCKED значение 1, когда в данный момент выполняется обновление. Если вы сделаете еще одно обновление в это время, Liquibase дождется снятия блокировки, прежде чем запускать его.

| **Column** | **Standard data type** | **Description** |
| --- | --- | --- |
| ID | INT | Идентификатор замка. В настоящее время существует только один замок. |
| LOCKED | INT | становите значение «1», если Liquibase работает с этой базой данных. В противном случае установите "0" |
| LOCKGRANTED | DATETIME | Дата и время предоставления блокировки. |
| LOCKEDBY | VARCHAR(255) | Удобочитаемое описание того, кому была предоставлена блокировка. |

# DATABASECHANGELOG

Liquibase использует таблицу DATABASECHANGELOG для отслеживания того, какие наборы изменений были запущены.

Таблица отслеживает каждый набор изменений как строку, определяемую комбинацией столбцов идентификатора, автора и имени файла.

В таблице нет первичного ключа. Это делается для того, чтобы избежать каких-либо ограничений длины ключа, связанных с базой данных. Состав идентификатора, автора и имени файла уникален для всех строк таблицы.

|  |  |  |
| --- | --- | --- |
| ID | VARCHAR(255) | Значение атрибута id  элемента changeset. |
| AUTHOR | VARCHAR(255) | Значение атрибута author  элемента changeset. |
| FILENAME | VARCHAR(255) | Путь к changelog. Это может быть абсолютный путь или относительный путь в зависимости от того, как changelog был передан в Liquibase . Для достижения наилучших результатов это должен быть относительный путь. Атрибут **[logicalFilePath](https://docs-liquibase-com.translate.goog/concepts/changelogs/attributes/logicalfilepath.html?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=wapp)** можно использовать в changelog или в отдельных changesets |
| DATEEXECUTED | DATETIME | Дата/время выполнения changeset . Используется с ORDEREXECUTED для определения порядка отката.. |
| ORDEREXECUTED | INT | Порядок выполнения changesets. Используется в дополнение к DATEEXECUTEDдля обеспечения правильного порядка, даже если дата и время базы данных поддерживают плохое разрешение.  **ПРИМЕЧАНИЕ.** Увеличение значений гарантировано только в рамках отдельного запуска обновления. Бывают случаи, когда они перезапускаются с нуля. |
| EXECTYPE | VARCHAR(10) | Описание того, как был выполнен changeset. Возможные значения включают EXECUTED, FAILED, SKIPPED, RERANи MARK\_RAN. |
| MD5SUM | VARCHAR(35) | Контрольная сумма набора изменений на момент его выполнения. Используется при каждом запуске, чтобы гарантировать отсутствие непредвиденных изменений  changesets в файле changelog. |
| DESCRIPTION | VARCHAR(255) | Краткое автоматически сгенерированное понятное человеку описание changeset |
| COMMENTS | VARCHAR(255) | Значение атрибута comment у changeset. |
| TAG | VARCHAR(255) | Отслеживает, какой changeset  соответствует операциям с тегами. |
| LIQUIBASE | VARCHAR(20) | Версия Liquibase, используемая для выполнения набора изменений |
| CONTEXTS | VARCHAR(255) | Контекст(ы), используемые для выполнения changeset. |
| LABELS | VARCHAR(255) | Метки, используемые для выполнения changeset. |
| DEPLOYMENT\_ID | VARCHAR(10) | changesets  развернутые вместе, будут иметь один и тот же уникальный идентификатор. |

# Лучшие практики

## Концепции Liquibase

Все изменения базы данных указаны в файле changelog  Liquibase  . Изменение содержится в changeset  , а changesets  добавляются в changelog  в том порядке, в котором они должны быть развернуты.

Проще говоря — changelog  содержит упорядоченный список changesets, а changeset содержит изменение .

форматы changelog: SQL, XML, JSON или YAML. И вы даже можете смешивать и сопоставлять различные типы changelogs, если хотите.

После того, как ваш changelog создан, запуск  liquibase update развертывает все неразвернутые изменения в целевой базе данных. Не волнуйтесь, Liquibase отслеживает, что развернуто, а что нет, в таблице DATABASECHANGELOG. Когда вы запускаете liquibase update команду, она обновляет целевую базу данных без необходимости вмешательства. Когда все изменения были развернуты, последующие вызовы liquibase updateуспешно возвращаются без каких-либо действий (поскольку делать нечего).

Следующие рекомендации содержат рекомендации по основным компонентам Liquibase и рабочему процессу.

## Организуйте свои журналы изменений

**1. Определите структуру каталогов**

Наиболее распространенный способ организации changelog — по основному выпуску.

Обязательно храните changelog в системе управления версиями, желательно рядом с кодом доступа к базе данных.

В этом примере мы будем использовать com/example/db/changelog.

com

example

db

changelog

db.changelog-root.xml

db.changelog-1.0.xml

db.changelog-1.1.xml

db.changelog-2.0.xml

DatabasePool.java

AbstractDAO.java

**2. Настройте корневой changelog  и включенные файлы(included) changelog.**

Файл db.changelog-root.xml будет автоматически включать **changelog** для каждого выпуска в алфавитно-цифровом порядке.

Примечание. db.changelog-root.xml Файл — это имя **changelog**, которое вы будете передавать всем вызовам Liquibase.

Для приведенного выше примера журнал изменений будет выглядеть следующим образом:

**<?xml version="1.0" encoding="UTF-8"?>**

<**databaseChangeLog**

**xmlns**="http://www.liquibase.org/xml/ns/dbchangelog"

**xmlns:xsi**="http://www.w3.org/2001/XMLSchema-instance"

**xmlns:pro**="http://www.liquibase.org/xml/ns/pro"

xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog

http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-4.1.xsd

http://www.liquibase.org/xml/ns/pro

http://www.liquibase.org/xml/ns/pro/liquibase-pro-4.1.xsd">

<**includeAll** **path**="com/example/db/changelog/"/>

</**databaseChangeLog**>

Включенные журналы изменений могут быть в любом формате (включая форматированный SQL). Выберите подходящий формат, который лучше всего подходит вам и ситуации вашей команды.

Используйте журнал изменений **SQL** , когда ваша команда

* уже знаком с SQL
* Хочет получить преимущества от использования Liquibase, не изучая новый способ указания изменений в базе данных.

Используйте журнал изменений **на основе модели** (XML, JSON или YAML), когда ваша команда

* Хочет развернуть единый набор изменений для разных типов баз данных.
* Хочет воспользоваться преимуществами автоматизации rollbackновых объектов базы данных
* Хочет воспользоваться Liquibase diffи diffChangelogфункциональностью

**changeset**

**1. Одно изменение на changeset**

Мы настоятельно рекомендуем вносить только одно изменение в **changeset**. Это делает каждое изменение атомарным в рамках одной транзакции. Каждый набор изменений либо завершается успешно, либо терпит неудачу. В случае сбоя его можно исправить и повторно развернуть до тех пор, пока он не добьется успеха. Множественные независимые изменения в одном наборе изменений создают риск того, что некоторые изменения будут развернуты, а более позднее изменение завершится ошибкой. Это оставляет базу данных в частично развернутом состоянии, которое требует ручного вмешательства для исправления.

Исключение составляют случаи, когда у вас есть несколько изменений, которые вы хотите сгруппировать как одну транзакцию — в этом случае несколько операторов в наборе изменений, если это правильный выбор.

**2. Определите формат идентификатора набора changeset.**

Выберите идентификатор набора изменений, который вам подходит . Хотя это может быть любая строка, мы рекомендуем использовать возрастающую числовую последовательность, начиная с 1. Помните, что каждый идентификатор набора изменений должен быть уникальным changelog.

**3. Документируйте неясные или сложные changesets**

В большинстве случаев **changeset**s являются самодокументируемыми.

Однако не забудьте использовать <comments> для любых **changeset**, когда вам нужно объяснить неочевидные или сложные изменения базы данных другим разработчикам.

**4. Имейте план отката**

Напишите **changeset**, чтобы они работали с [откатом Liquibase](https://translate.google.com/website?sl=en&tl=ru&hl=ru&client=webapp&u=https://docs.liquibase.com/workflows/liquibase-community/using-rollback.html) .

* Используйте соответствующий [тип изменения Liquibase](https://translate.google.com/website?sl=en&tl=ru&hl=ru&client=webapp&u=https://docs.liquibase.com/change-types/home.html) вместо использования пользовательского <sql>тега.
* Включайте [тег отката Liquibase](https://translate.google.com/website?sl=en&tl=ru&hl=ru&client=webapp&u=https://docs.liquibase.com/commands/community/rollbackbytag.html) ( <rollback>) всякий раз, когда изменение не поддерживает готовый откат. (например, <sql>, <insert>, и т.д.).

Обязательно тестируйте откаты в процессе разработки, чтобы гарантировать безопасность и предсказуемость отката в рабочей среде.

**5. Управляйте своими справочными данными**

Используйте Liquibase для управления справочными данными. Разделение среды (DEV, QA, PROD) может быть достигнуто с помощью [контекстов Liquibase](https://translate.google.com/website?sl=en&tl=ru&hl=ru&client=webapp&u=https://docs.liquibase.com/concepts/contexts.html) . Эта функция полезна в следующих ситуациях:

* Когда у вас есть тестовые данные, которые следует развертывать только в средах контроля качества
* При управлении данными конфигурации приложения — таблицей стран, данными конфигурации приложения и т. д.
* При развертывании исправлений данных, характерных для предпроизводственной и производственной сред.

## Управление хранимой логикой

Обычно **changeset** в Liquibase остаются нетронутыми после их развертывания. Последующие изменения базы данных вносятся, когда в конец changelog добавляются новые **changeset** с повтором транзакций .

Однако хранимая логика (хранимые процедуры, функции, пакеты, триггеры и т. д.) действует больше как код приложения, чем изменения схемы базы данных. Они лучше управляются подобно исходному коду, где вы постоянно обновляете один исходный файл для каждой единицы хранимой логики. Это также позволяет вам лучше видеть изменения с течением времени, используя стандартные инструменты git.

Мы рекомендуем вести отдельный changelog , посвященный хранимой логике. Комбинация «создать или заменить» в сохраненной логике sql вместе с runOnChange="true"атрибутом **changeset** позволяет гарантировать, что изменения будут развернуты тогда и только тогда, когда изменится файл. В частности, установка этого флага заставляет Liquibase проверять, был ли изменен набор изменений. Обнаружив модификацию, Liquibase развертывает (обновленное) изменение.

## Разрабатывайте программное обеспечение, используя этот стандартный рабочий процесс

1. Используя вашу любимую IDE или редактор, создайте новый локальный changeset, содержащий изменение

2. Запустите liquibase update, чтобы выполнить новый changeset

3. Внесите соответствующие изменения в код приложения

4. Протестируйте новый код приложения вместе с изменением базы данных.

5. Зафиксируйте как changeset , так и код приложения в системе управления версиями .

## Что дальше?

### Используйте Liquibase для достижения CI/CD для баз данных

Миграция схемы базы данных является важной задачей для каждого программного проекта. [Узнайте, как интегрировать Liquibase в ваш процесс для достижения CI/CD для баз данных](https://translate.google.com/website?sl=en&tl=ru&hl=ru&client=webapp&u=https://www.liquibase.com/ci-cd) .

### Рассмотрите расширенные функции и поддержку Liquibase

Расширенные функции и поддержка, предназначенные для команд любого размера, доступны через [Liquibase.com](https://translate.google.com/website?sl=en&tl=ru&hl=ru&client=webapp&u=https://www.liquibase.com) .

Что такое непрерывная интеграция (CI)?

**Непрерывная интеграция (**CI) — это процесс, который автоматизирует интеграцию изменений кода от нескольких разработчиков в один программный проект. Процесс CI зависит от системы контроля версий исходного кода, такой как Git (или Github, BitBucket и многие другие) или контроля версий TFS, в сочетании с автоматизированными процессами проверки качества кода, инструментами проверки стиля синтаксиса и т. д. Эти процессы могут запускаться, когда новый код объединяется или фиксируется в общем репозитории, отсюда и название «Непрерывная интеграция».

## CI/CD для баз данных с использованием Liquibase

Внедрение сквозной CI/CD требует, чтобы весь код (включая код базы данных) был проверен в системе контроля версий и развернут как часть процесса выпуска программного обеспечения. Liquibase может помочь вам в этом.

Каждое изменение схемы базы данных, которое вы делаете с помощью Liquibase, называется changeset . Все наборы изменений отслеживаются Liquibase с помощью changelogs Liquibase позволяет вам создать триггер, который автоматически обновляет базу данных, указывая на файл журнала изменений. Отсюда можно легко интегрировать процесс в общий процесс CI/CD:

* Отправьте файлы changeset  в репозиторий функций.
* Создайте запрос на извлечение для ветки Dev
* После экспертной оценки и утверждения объедините функциональную ветку с веткой Dev.
* Реализация CI/CD, настроенная на сервере разработки, запускает Liquibase для обновлений базы данных.
* Liquibase автоматически выполняет любые новые файлы changelog  (и достаточно удобен, чтобы помнить, какие сценарии уже запущены)

# liquibase.properties

changeLogFile= changelog/db.changelog-master.xml  
url= jdbc:mysql://localhost:3306/sso?createDatabaseIfNotExist=true&allowPublicKeyRetrieval=true&useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC  
username= root  
password=root  
driver= com.mysql.cj.jdbc.Driver

**changelog/** [db.changelog-master.xml](https://bitbucket.org/artezio_amoskalenko/artezio-sso/src/master/src/main/resources/changelog/db.changelog-master.xml)

<?xml version="1.0" encoding="UTF-8"?>  
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog  
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">  
  
<preConditions>  
<dbms type="mysql"/>  
</preConditions>  
  
<changeSet id="1" author="Alena.M">  
<tagDatabase tag="1.0"/>  
</changeSet>  
  
<include file="v-1.0/db.changelog-v.1.0.xml" relativeToChangelogFile="true"/>  
  
</databaseChangeLog>

**changelog/** [v-1.0](https://bitbucket.org/artezio_amoskalenko/artezio-sso/src/master/src/main/resources/changelog/v-1.0/)/**db.changelog-v.1.0.xml**

<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog  
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">  
  
<include file="1-ChangsetUsersTable.xml" relativeToChangelogFile="true"/>  
<include file="2-ChangsetRolesTable.xml" relativeToChangelogFile="true"/>  
<include file="3-ChangsetApplicationsTable.xml" relativeToChangelogFile="true"/>  
<include file="4-ChangsetUsersRolesAppsTable.xml" relativeToChangelogFile="true"/>  
<include file="5-ChangsetSessionsTable.xml" relativeToChangelogFile="true"/>  
  
</databaseChangeLog>

**changelog/** [v-1.0](https://bitbucket.org/artezio_amoskalenko/artezio-sso/src/master/src/main/resources/changelog/v-1.0/)/**1-ChangsetUsersTable.xml**

<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog  
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">  
  
  
<changeSet id="1.1" author="Alena.M">  
<sqlFile dbms="mysql"  
encoding="UTF-8"  
endDelimiter="\nGO"  
relativeToChangelogFile="true"  
splitStatements="true"  
stripComments="true"  
path="1-CreateUsersTable.sql"/>  
<rollback>  
<sqlFile dbms="mysql"  
encoding="UTF-8"  
endDelimiter="\nGO"  
relativeToChangelogFile="true"  
splitStatements="true"  
stripComments="true"  
path="1-DropUsersTable.sql"/>  
</rollback>  
  
</changeSet>  
  
</databaseChangeLog>

**changelog/** [v-1.0](https://bitbucket.org/artezio_amoskalenko/artezio-sso/src/master/src/main/resources/changelog/v-1.0/)/1-CreateUsersTable.sql

CREATE TABLE Users (  
id int(11) AUTO\_INCREMENT,  
login VARCHAR(80),  
password VARCHAR(255),  
PRIMARY KEY (id)  
);

**changelog/** [v-1.0](https://bitbucket.org/artezio_amoskalenko/artezio-sso/src/master/src/main/resources/changelog/v-1.0/)/1-DropUsersTable.sql

DROP TABLE Users;

* [Preconditions](http://www.liquibase.org/documentation/preconditions.html) — позволяют задавать условие выполнения changeset-a;

Иногда перед применением изменений нужно проверить некоторые условия, например, проверить, что таблица пустая, перед её удалением. Для этого используется тег preConditions.

## Использование

Вы можете использовать предварительные условия для:

* Задокументируйте, какие предположения были у автора журнала изменений при его создании.
* Обеспечьте, чтобы эти предположения не нарушались пользователями, работающими с журналом изменений .
* Выполняйте проверки данных перед внесением необратимых изменений, таких как **[dropTable](https://docs-liquibase-com.translate.goog/change-types/drop-table.html?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=wapp)** .
* Контролируйте, какие наборы изменений выполняются, а какие нет, в зависимости от состояния базы данных.

Вы можете использовать все предварительные условия Liquibase в журналах изменений XML, YAML и JSON  . Единственным поддерживаемым предварительным условием для журналов  изменений SQL является . Список предварительных условий см . в разделе [**Доступные предварительные условия**](https://docs-liquibase-com.translate.goog/concepts/changelogs/preconditions.html?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=wapp#available-preconditions) .sqlCheck

## Запуск журналов изменений с предварительными условиями

В журнале изменений или наборе изменений может быть только один элемент предварительных условий . Если вам нужно несколько предварительных условий , вы можете использовать вложенные предварительные условия . Кроме того, предварительные условия на уровне журнала изменений применяются ко всем наборам изменений , а не только к тем, которые перечислены в текущем журнале изменений или его дочерних журналах изменений .

Запустив журнал изменений или набор изменений с предварительным условием, добавьте его в свой файл журнала изменений , как показано в примерах.

[**[Открытым](https://docs-liquibase-com.translate.goog/concepts/changelogs/preconditions.html?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=wapp)XML-пример**](https://docs-liquibase-com.translate.goog/concepts/changelogs/preconditions.html?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=wapp)

<?xml version="1.1" encoding="UTF-8" standalone="no"?>

<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ext="http://www.liquibase.org/xml/ns/dbchangelog-ext" xmlns:pro="http://www.liquibase.org/xml/ns/pro" xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-latest.xsd http://www.liquibase.org/xml/ns/dbchangelog-ext http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-ext.xsd http://www.liquibase.org/xml/ns/pro http://www.liquibase.org/xml/ns/pro/liquibase-pro-latest.xsd">

<preConditions>

<dbms type="oracle" />

<runningAs username="SYSTEM" />

</preConditions>

<changeSet id="1" author="Liquibase User">

<preConditions onFail="WARN">

<sqlCheck expectedResult="0">select count(\*) from oldtable</sqlCheck>

</preConditions>

<comment>Comments should go after preCondition. If they are located before the precondition, then Liquibase usually gives error.</comment>

<dropTable tableName="oldtable"/>

</changeSet>

</databaseChangeLog>

Пример журнала изменений будет запущен только в том случае, если исполняемая база данных — Oracle, а пользователь базы данных, выполняющий сценарий, — SYSTEM. Кроме того, он запустит набор изменений с sqlCheckпредварительным условием и **[dropTable](https://docs-liquibase-com.translate.goog/change-types/drop-table.html?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=wapp)** .

Если проверка предварительных условий не пройдена, вы получите предупреждение, и он продолжит выполнять набор изменений как обычно из-за onFail="WARN"предварительного условия. Чтобы предотвратить выполнение набора изменений при сбое предварительного условия, вы можете установить HALTили CONTINUEзначения. Дополнительные сведения см [**. в разделе значения onFail/onError**](https://docs-liquibase-com.translate.goog/concepts/changelogs/preconditions.html?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=wapp#onFail/onError) .

## Обработка сбоев и ошибок

Liquibase определяет два типа предварительных условий :

* Ошибки предусловия, которые означают, что проверка не удалась
* Ошибки предварительного условия, которые являются исключениями, возникающими при выполнении проверки.

Оба процесса можно контролировать с помощью атрибутов иonFail тега .onError <preConditions>

### Доступные атрибуты \_

| **Атрибут** | **Описание** |
| --- | --- |
| onFail | Контролирует, что произойдет, если проверка предварительных условий не пройдена. |
| onError | Управляет тем, что происходит, если возникает ошибка при проверке того, прошло ли предусловие или нет. |
| onSqlOutput | Контролирует, что делать в update-sqlрежиме. **С 1.9.5** |
| onFailMessage | Предоставляет пользовательское сообщение для вывода, когда предварительные условия не выполняются. **Начиная с версии 2.0** |
| onErrorMessage | Предоставляет пользовательское сообщение для вывода, когда предварительные условия не выполняются. **Начиная с версии 2.0** |

### onFail/ onErrorзначения

| **Ценность** | **Описание** |
| --- | --- |
| HALT | Останавливает выполнение всего журнала изменений (по умолчанию). HALTможет быть помещен вне набора изменений (например, в начале журнала изменений ). |
| CONTINUE | Пропускает набор изменений . Выполнение набора изменений будет предпринято снова при следующем обновлении. Продолжение журнала изменений . |
| MARK\_RAN | Пропускает набор изменений , но помечает его как выполненный. Продолжение журнала изменений . |
| WARN | Отправляет предупреждение и продолжает выполнение набора изменений / журнала изменений в обычном режиме. WARNможет быть помещен вне набора изменений (например, в начале журнала изменений ). |

### onSqlOutputценности

| **Ценность** | **Описание** |
| --- | --- |
| TEST | Запускает набор изменений в update-sqlрежиме. |
| FAIL | Сбой preConditionв update-sqlрежиме. |
| IGNORE | Игнорирует preConditionв update-sqlрежиме (по умолчанию). |

## Доступные предварительные условия

Примеры предусловий , показанные в таблице, не включают schemaName атрибут . Лучше всего не сохранять schemaNameв наборе изменений и полагаться на схему по умолчанию, за исключением случаев, когда вы не хотите создавать что-либо в схеме по умолчанию.

Кроме того, вы можете включить атрибутыonError или со значением , , или в тег предварительных условий , однако параметры и можно применять только к предварительным условиям внутри набора изменений .onFail WARNHALTCONTINUEMARK\_RANCONTINUEMARK\_RAN

Вы можете использовать любое предварительное условие, указанное в следующей таблице:

|  |  |  |
| --- | --- | --- |
| **предварительное условие** | **Описание** | **Атрибут** |
| dbms | Определяет, соответствует ли выполняемая база данных указанному типу.  <preConditions onFail="WARN">  <or>  <dbms type="oracle" />  <dbms type="mysql" />  </or>  </preConditions> | **type** – ожидаемый тип базы данных. Можно указать несколько значений dbms, используя значения, разделенные запятыми. (требуется) |
| runningAs | Определяет, соответствует ли пользователь базы данных, исполняемый под именем, указанному имени пользователя.  <preConditions onFail="WARN">  <dbms type="oracle" />  <runningAs username="SYSTEM" />  </preConditions> | **имя пользователя** — пользовательский скрипт базы данных, который должен запускаться от имени. (требуется) |
| changeSetExecuted | Определяет, был ли уже выполнен указанный набор изменений .  <preConditions onFail="HALT">  <changeSetExecuted id="1" author="liquibase" changelog-file="changelog.xml" />  </preConditions> | * **Id** — идентификатор набора изменений . (требуется) * **author** – автор набора изменений . (требуется) * **changelog-file** — имя файла (включая относительный путь к classpath) набора изменений . (требуется) |
| columnExists | Определяет, существует ли указанный столбец в базе данных.  <preConditions onFail="WARN">  <columnExists tableName="table1" columnName="column1" />  </preConditions> | * **schemaName** – имя схемы таблицы. * **tableName** – имя таблицы столбца. (требуется) * **columnName** — имя столбца. (требуется) |
| tableExists | Определяет, существует ли указанная таблица в базе данных.  <changeSet author="author" id="1">  <preConditions onFail="MARK\_RAN">  <not>  <tableExists tableName="MY\_NEW\_TABLE"/>  </not>  </preConditions>  <createTable tableName="MY\_NEW\_TABLE">  <column name="IDX" type="${integer.type}">  <constraints nullable="false"/>  </column>  <column name="INTVAL" type="${integer.type}"/>  </createTable>  </changeSet> | * **schemaName** – имя схемы таблицы. * **tableName** — имя таблицы. (требуется) |
| viewExists | Определяет, существует ли указанное представление в базе данных.  <preConditions onFail="HALT">  <viewExists viewName="liquibase" />  </preConditions> | * **schemaName** — имя схемы представления. * **viewName** — имя представления. (требуется) |
| foreignKeyConstraintExists | Определяет, существует ли указанный внешний ключ в базе данных.  <preConditions onFail="WARN">  <foreignKeyConstraintExists foreignKeyName="PK\_2354" />  </preConditions> | * **schemaName** — имя схемы внешнего ключа. * **externalKeyName** — имя внешнего ключа. (требуется) |
| indexExists | Определяет, существует ли указанный индекс в базе данных. Вы можете либо указать indexName атрибутtableName , либо columnNames атрибут s.  <preConditions onFail="WARN">  <indexExists indexName="originindex"/>  </preConditions>  <preConditions onFail="WARN">  <indexExists tableName="MY\_NEW\_TABLE" columnNames="ID"/>  </preConditions>  **Примечание.** Есть несколько баз данных, вindexNameкоторых не является уникальным, поэтомуможно использоватьindexNameиtableName  <preConditions onFail="WARN">  <indexExists indexName="originindex" tableName="MY\_NEW\_TABLE"/>  </preConditions> | * **schemaName** — имя схемы индекса. * **indexName** — имя индекса. * **tableName** — имя таблицы. * **columnNames** — имя столбца. |
| sequenceExists | Определяет, существует ли указанная последовательность в базе данных.  <preConditions onFail="WARN">  <sequenceExists sequenceName="idnumber" />  </preConditions> | * **schemaName** – имя схемы последовательностей. * **sequenceName** — имя последовательности. (требуется) |
| primaryKeyExists | Определяет, существует ли указанный первичный ключ в базе данных.  <preConditions onFail="HALT">  <primaryKeyExists primaryKeyName="ID\_2354" tableName="table1" />  </preConditions> | * **schemaName** — имя схемы первичного ключа. * **primaryKeyName** — имя ограничения первичного ключа. * **tableName** — имя таблицы, содержащей первичный ключ. ( tableNameили primaryKeyNameтребуется) |
| sqlCheck | Выполняет строку SQL и проверяет возвращаемое значение. SQL должен возвращать одну строку с одним значением.   * Чтобы проверить количество строк, используйте countфункцию SQL. * Чтобы проверить диапазоны значений, выполните проверку в SQL и верните значение, с которым можно легко сравнить.   <preConditions onFail="WARN">  <sqlCheck expectedResult="1">  SELECT COUNT(1) FROM pg\_tables WHERE TABLENAME = 'myRequiredTable'  </sqlCheck>  </preConditions> | **ожидаемый** результат — значение, с которым сравнивается результат SQL. (требуется) |
| changeLogPropertyDefined | Проверяет наличие данного атрибута журнала изменений . Это не удается, если значение не совпадает с заданным.  <preConditions onFail="WARN">  <changeLogPropertyDefined property="myproperty"/>  <changeLogPropertyDefined property="myproperty" value="requiredvalue"/>  </preConditions> | * **property** — имя свойства для проверки. (требуется) * **value** – требуемое значение для данного свойства. |
| customPrecondition | Может быть создан путем добавления класса, реализующего liquibase.precondition.CustomPreconditionинтерфейс. Параметры пользовательских классов задаются посредством отражения на основе <param>вложенных тегов. Передайте параметры в виде строк пользовательскому предварительному условию.  <customPrecondition className="com.example.CustomTableCheck">  <param name="tableName" value="our\_table"/>  <param name="count" value="42"/>  </customPrecondition> | **className** — имя пользовательского класса преусловия. (требуется)  **Вложенные теги customPrecondition :**   * **param** — параметр для передачи пользовательскому предварительному условию. * **name** – имя устанавливаемого параметра. (требуется) * **value** – строковое значение для установки параметра. (требуется) |
| uniqueConstraintExists | Проверяет наличие уникальных ограничений перед запуском обновления. (начиная с Liquibase 4.9.0)  <preConditions>  <uniqueConstraintExists constraintName="uq\_constraint\_name" tableName="test\_table"/>  </preConditions>  <preConditions>  <uniqueConstraintExists tableName="test\_table" columnNames="first\_name, last\_name"/>  </preConditions> | * **limitedName** — имя уникального ограничения. * **tableName** – имя таблицы столбца (обязательно) * **columnNames** — имя столбца |