
# Examples:

# groovy-liquibase
# Tim Berglund

# Groovy Migrations

This document contains all liquibase changes translated to groovy

**Created by:** Erwin van Brandwijk intern of 42bv
**Website:** www.42.nl
**Date:** 15 – February – 2011

# Liquibase database changes

This document contains all changes from Liquibase that can be executed with the grails-plugin database-migration. These are the examples from liquibase.org/manuel. I have converted them to groovy changes. After that I tested them on PostgreSQL and some on Mysql.

### Example: changelogFile

```
databaseChangeLog() {
	// changeSets
}
```

If quotes are empty, you do not have to use that parameter. See example with addColumn - schemaName.

### Example: Add Column

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	addColumn(tableName: "books", schemaName: "") {
		column(name: "firstname", type: "varchar(255)")
	}
}
```

Could be ...

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	addColumn(tableName: "person") {
		column(name: "firstname", type: "varchar(255)")
	}
}
```

If the comment tag contains rollback = no, you will have to create your own rollback. You have got two possibilities. See next example....

### Example: Drop Column with own rollback

```
changeSet(author: "erwin", id: "4") {
		comment("rollback = no")

		dropColumn(tableName: "Person", columnName: "test")

		rollback{
		addColumn(tableName: "Person") {
			column(name: "test", type: "varchar(255)")
		}
		//of SQL in rollback
		rollback(“””
			//SQL QUERY;
		“””)
	}
```

Do Not use groovy outside a grails change, this will always be executed!! Unless you want that.
When using a Boolean, don't use quotes example: `nullable: false`

# Liquibase changes

## Structural Refactorings

http://www.liquibase.org/manual/home

### Add column

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	addColumn(tableName: "persons", schemaName: "") {
		column(name: "boss", type: "int") {
			constraints(nullable: false, foreignKeyName:
			"FK_persons_boss_persons_id", references: "persons(id)")
		}
	}
	//	possible parameters → Column
	// String name, String type, String value, Number valueNumeric, Date valueDate,
	// Boolean valueBoolean, String defaultValue, Number defaultValueNumeric,
	// Date defaultValueDate, Boolean defaultValueBoolean,
	// DatabaseFunction defaultValueComputed, Boolean autoIncrement, String remarks

	//	possible parameters → Constraints
	// Boolean nullable, Boolean primaryKey, String primaryKeyName,
	// String primaryKeyTablespace, String references, Boolean unique,
	// String uniqueConstraintName, String check, Boolean deleteCascade,
	// String foreignKeyName, Boolean initiallyDeferred, Boolean deferrable
}
```

### Rename column

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	renameColumn(tableName: "test", oldColumnName: "testcolumn", schemaName: ""
		newColumnName: "testrenamecolumn", columnDataType: "int")
}
```

### Drop column

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	dropColumn(tableName: "person", schemaName: "", columnName: "testcolumn")
}
```

### Alter sequence

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	alterSequence(sequenceName: "seq_employee_id", incrementBy: "10")
}
```

### Create Table

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	createTable(tableName: "person", schemaName: "", tablespace: "", remarks: "") {
		column(name: "id", type: "int") {
			constraints(primaryKey: true, nullable: false)
		}
		column(name: "firstname", type: "varchar(255)")
		column(name: "lastname", type: "varchar(255)")
		column(name: "username", type: "varchar(255)") {
			constraints(unique: true, nullable: false)
		}
		column(name: "testid", type: "int")
	}
}
```

### Rename table

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	renameTable(oldTableName: "employee", schemaName: "", newTableName: "person")
}
```

### Drop table

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	dropTable(tableName: "person", schemaName: "mySchema", cascadeConstraints: true)
	// @ cascadeConstraints
	// Mysql won't work, before	->	sql("SET foreign_key_checks = 0;")
	// After dropTable		->	sql("SET foreign_key_checks = 1;")
}
```

### Create view

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	createView(viewName: "personView", schemaName: "", replaceIfExists: true) {
		"select id, firstname from person"
	}
}
```

### Rename view

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	renameView(schemaName: "", oldViewName: "personView", newViewName: "people")
}
```

### Drop view

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	dropView(schemaName: "", viewName: "personView")
}
```

### Merge columns

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	mergeColumns(tableName: "person", schemaName: "", column1Name: "phoneAreaCode",
		joinString: "-", column2Name: "phoneSuffix", finalColumnName: "phone",
		finalColumnType: "varchar(50)")
}
```

### Create stored procedure

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	createStoredProcedure("""
		CREATE OR REPLACE PROCEDURE testHello
			IS
			BEGIN
			  DBMS_OUTPUT.PUT_LINE('Hello From The Database!');
			END;""")
}
```

## Data quality refactorings

### Add lookup table

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	addLookupTable(existingTableName: "address",
			existingColumnName: "state",
			existingTableSchemaName: "",
			newTableSchemaName: "",
			newTableName: "state",
			newColumnName: "abbreviation",
			constraintName: "fk_address_state",
			newColumnDataType: "Required for mysql")
}
```

### Add not-null constraint

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	addNotNullConstraint(tableName: "employee", schemaName: "",
		columnName: "employer_id", defaultNullValue: "1",
		columnDataType: "required for mysql and ms-sql only")
}
```

### Drop not-null constraint

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	dropNotNullConstraint(tableName: "employee", schemaName: "", columnName: "id",
		columnDataType: "Required mysql, ms-sql and postrgreSQL")
}
```

### Add unique constraint

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	addUniqueConstraint(tableName: "person", schemaName: "",
		tablespace: "", columnNames: "id", constraintName: "pk_person",
		deferrable: "", initiallyDeferred: "", diabled: "")
}
```

### Drop unique constraint

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	dropUniqueConstraint(tableName: "person",
		schemaname: "", constraintName: "pk_person")
}
```

### Create sequence

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	createSequence(sequenceName: "seq_employee_id", schemaName: "",
		incrementBy: "", minValue: "", maxValue: "", startValue: "",
		ordered: "", cycle: "")
}
```

### Drop sequence

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	dropSequence(sequenceName: "seq_employee_id", schemaName: "")
}
```

### Add auto-increment

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	addAutoIncrement(tableName: "person", schemaName: "",
		columnName: "id", columnDataType: "int")
}
```

### Add default value

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes, one of defaultValue* required")

	addDefaultValue(tableName: "file", schemaName: "", columnName: "fileName",
		columnDataType: “”, defaultValue: "New File", defaultValueNumeric: "1",
		defaultValueBoolean: true, defaultValueDate: "")
}
```

### Drop default value

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	dropDefaultValue(tableName: "file", schemaName: "",
		columnName: "fileName", dataColumnType: "")
}
```

## Referential integrity refactorings

### Add foreign key constraint

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	addForeignKeyConstraint(constraintName: "fk_address_person",
		baseTableName: "address", referencedTableName: "person",
		baseTableSchemaName: "", referencedTableSchemaName: "",
		baseColumnNames: "person_id",  referencedColumnNames: "id",
		deferrable: "", initiallyDeffered: "", onDelete: "", onUpdate: "")
}
```

### Drop foreign key constraint

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	dropForeignKeyConstraint(constraintName: "fk_address_person",
		baseTableName: "address", baseTableSchemaName: "")
}
```

### Drop all foreign key constraints

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	dropAllForeignKeyConstraints(baseTableName: "address", baseTableSchemaName: "")
}
```

### Add primary key constraint

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	addPrimaryKey(tableName: "person", schemaName: "", tablespace: "",
		columnNames: "id", constraintName: "pk_person")
}
```

### Drop primary key constraint

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	dropPrimaryKey(tableName: "person", schemaName: "", constraintName: "pk_person")
}
```

## Non-Refactoring transformations

### Insert data

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	insert(tableName: "People", schemaName: "") {
		column(name: "id", valueNumeric: "2")
		column(name: "firstname", value: "Fred")
		column(name: "testid", valueNumeric: "2")
	}
}
```

### Load data

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	loadData(tableName: "users", schemaName: "", file: "com/sample/users.csv",
		encoding: "") {
		column(name: "id", type: "NUMERIC")
		column(name: "firstname", type: "STRING")
	}
}
```

### Load update data

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no, only Oracle")

	loadUpdateData(tableName: "users", schemaName: "", file: "com/sample/users.csv",
		encoding: "", primaryKey: "id") {
		column(name: "id", type: "NUMERIC")
		column(name: "firstname", type: "STRING")
	}
}
```

### Update data

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	update(tableName: "ProductSettings", schemaName: "") {
		column(name: "property", value: "vatCategory")
		where("property='vat'")
	}
}
```

### Delete data

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	delete(tableName: "People", schemaName: "") {
		where("id=2")
	}
}
```

### Tag database

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	tagDatabase(tag: "version_1.3")
}
```

### Stop

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	stop("Halted LiquiBase for debugging")
}
```

## Architectural refactorings

### Create index

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes")

	createIndex(tableName: "user", schemaName: "", indexName: "idx_person_name",
		tablespace: "", unique: “”) {
		column(name: "firstname")
		column(name: "lastname")
	}
}
```

### Drop index

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	dropIndex(indexName: "idx_user_username", tableName: "table_name", schemaName: "")
}
```

## Custom refactorings

### Modifying generated SQL

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no, more info on liquibase.org")

	createTable(tableName: "person") {
		column(name: "id", type: "bigint")
		column(name: "firstname", type: "varchar(255)")
		column(name: "lastname", type: "varchar(255)")
	}
	modifySql {
		replace(replace: "bigint", with: "long")
	}
	modifySql(dbms: "mysql") {
		append(value: " engine innodb") // prepend(value: "sql")   add before SQL
	}
}
```

### Custom SQL

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	sql("insert into person (id, name) values (1, 'Bob')")
	sql(stripComments: true, splitStatements: false, endDelimiter: ';') {
	"insert into person (id, name) values (1, 'Bob')"
	}
}
```

### Custom SQL file

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	sqlFile(path: "sample.sql", stripComments: true or false,
		splitStatements: true or false, encoding: "",
		endDelimiter: "; or / or something else",
		relativeToChangelogFile: true or false)
}
```

### Custom refactoring class

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = yes if made, more info on liquibase.org")

	customChange(class: 'com.augusttechgroup.liquibase.MonkeyRefactoring') {
		tableName('animal') // params from custom class
		species('monkey')
		status('angry')
	}
}
```

### Execute shell command

```
changeSet(author: "erwin", id: "1") {
	comment("rollback = no")

	executeCommand(executable: 'gedit', os: 'unix') {
		arg('changelog.groovy')
	}
}
```

## Other

### Preconditions (changelog)

```
databaseChangeLog() {

  preConditions(onFail: 'WARN') {
//     possible parameters → preconditions
// onError:         “HALT/WARN/MARK_RAN/CONTINUE”
// onUpdateSql:     “RUN/FAIL/IGNORE”
// onFailMessage:, onErrorMessage:  “message string”

    and {
      dbms(type: 'mysql')
      runningAs(username: 'root')
      or {
        changeSetExecuted(id: '', author: '', changeLogFile: '')
        columnExists(schemaName: '', tableName: '', columnName: '')
        tableExists(schemaName: '', tableName: '')
        viewExists(schemaName: '', viewName: '')
        foreignKeyConstraintExists(schemaName: '', foreignKeyName: '')
        indexExists(schemaName: '', indexName: '')
        sequenceExists(schemaName: '', sequenceName: '')
        primaryKeyExists(schemaName: '', primaryKeyName: '', tableName: '')
        sqlCheck(expectedResult: '') {
          "SELECT COUNT(1) FROM monkey WHERE status='angry'"
        }
        customPrecondition(className: '') {
          tableName('our_table')
          count(42)
        }
      }
    }
  }
}
```

### Preconditions (changeset)

```
// preconditions before comment!

changeSet(author: "erwin", id: "1") {
	preConditions(onFail: "HALT") {
		sqlCheck("select count(*) from persoon", expectedResult: "1")
	}
	comment("If precondition fails, stop")
}

//	possible parameters → preconditions
// onError:		“HALT/WARN/MARK_RAN/CONTINUE”
// onUpdateSql:	“RUN/FAIL/IGNORE”
// onFailMessage:, onErrorMessage:  “message string”

// It is also possible to use "and" and "or

changeSet(author: "erwin", id: "1") {
	preConditions (onFail: “WARN”, onFailMessage: "changeset 1 failed") {
		or {
			and {
				dbms(type: "oracle")
				runningAs(username: "SYSTEM")
			}
			and {
				dbms(type: "mysql")
				runningAs(username: "as")
			}
		}
	}
	comment("If precondition fails, warning")
}
```

### Contexts

```
// You can use context to specify which changeSets to be run.
// grails dbm-update --contexts=test
// Only changeSets with context: “test” or 'no context set' will run.
// Multiple contexts possible. See example

changeSet(author: "erwin", id: "1", context: "test","production")
	comment("comments")
}
```

### Include

```
// If you want to include other changelog(s).groovy for better structure use include
// Include is outside a changeSet. Use file:

databaseChangeLog() {
	include(file: "file.groovy")
}

// If you want to include all changelogs from a path. Use path:

databaseChangeLog() {
	include(path: "path/")
}
```