

# Living Documentation

Version 1.0.0-RC3-SNAPSHOT

# Table of Contents

<b>1. Introduction</b>	1
<b>2. Seeding database</b>	2
2.1. Seed database in a JUnit 4 test	2
<b>3. Manage database with Database Rider CDI</b>	5
3.1. Seed database using yaml dataset	5
<b>4. Manage database with Database Rider Cucumber</b>	9
4.1. Seed database using Database Rider in Cucumber tests	11
<b>5. Manage database with Database Rider and JUnit 5</b>	16
5.1. Seed database using Database Rider in JUnit5 tests	16
<b>6. Dynamic data using scritable datasets</b>	18
6.1. Seed database with groovy script in dataset	18
6.2. Seed database with javascript in dataset	19
<b>7. Database assertion using expected datasets</b>	21
7.1. Database assertion with yaml dataset	21
7.2. Database assertion with regular expression in expected dataset	22
7.3. Database assertion with seeding before test execution	23
7.4. Failing database assertion	25
7.5. Database assertion using automatic transaction	26

# Chapter 1. Introduction

**Database Rider** aims for bringing [DBUnit](#) closer to your JUnit tests so **database testing will feel like a breeze!**. Here are the main features:

- [JUnit rule](#) to integrate with DBUnit via annotations:

```
@Rule
public DBUnitRule dbUnitRule = DBUnitRule.instance(jdbcConnection);①

@Test
@DataSet(value = "datasets/yml/users.yml")
public void shouldSeedDataSet(){
    //database is seed with users.yml dataset
}
```

① The rule depends on a JDBC connection.

- [CDI integration](#) via interceptor to seed database without rule instantiation;
- JSON, YAML, XML, XLS, and CSV support;
- [Configuration](#) via annotations or yml files;
- [Cucumber](#) integration;
- Multiple database support;
- Date/time support in datasets;
- Scriptable datasets with groovy and javascript;
- Regular expressions in expected datasets;
- [JUnit 5](#) integration;
- [DataSet export](#);
- [Connection leak detection](#);
- Lot of [examples](#).

The project is composed by 5 modules:

- [Core](#): Contains the dataset executor and JUnit rule;
- [CDI](#): provides the DBUnit interceptor;
- [Cucumber](#): a CDI aware cucumber runner;
- [JUnit5](#): Comes with an [extension](#) for JUnit5.
- [Examples module](#).

# Chapter 2. Seeding database

In order to insert data into database  
As a developer  
I want to easily use DBUnit in JUnit tests.

Database Rider brings [DBUnit](#) to your [JUnit tests](#) via [JUnit rules](#) (in JUnit4), through a [CDI interceptor](#) (in [CDI](#) based tests) or by an [extension](#) (in case of [JUnit5](#) tests).

## 2.1. Seed database in a JUnit 4 test

JUnit4 is integrated with DBUnit by means of a [JUnit rule](#) called [DBUnitRule](#) which just needs a [JDBC](#) connection in order to be created.

This rule just reads [@Dataset](#) annotation in order to prepare the database state using DBUnit behind the scenes.

## Dependencies

To use it just add the following maven dependency:

```
<dependency>
  <groupId>com.github.database-rider</groupId>
  <artifactId>rider-core</artifactId>
  <version>1.0.0-RC3-SNAPSHOT</version>
  <scope>test</scope>
</dependency>
```

*Given*

The following junit rules 🍌

```
@RunWith(JUnit4.class)
public class DatabaseRiderIt {

    @Rule
    public EntityManagerProvider emProvider =
        EntityManagerProvider.instance("rules-it"); ①

    @Rule
    public DBUnitRule dbUnitRule =
        DBUnitRule.instance(emProvider.connection()); ②
}
```

- ① **EntityManagerProvider** is a simple Junit rule that creates a JPA entityManager for each test. DBUnit rule don't depend on EntityManagerProvider, it only needs a **JDBC connection**.
- ② DBUnit rule responsible for reading **@DataSet** annotation and prepare the database for each test.

And

The following dataset 🍌

*src/test/resources/dataset/yml/users.yml*

```
user:
  - id: 1
    name: "@realpestando"
  - id: 2
    name: "@dbunit"
tweet:
  - id: abcdef12345
    content: "dbunit rules!"
    date: "[DAY,NOW]"
    user_id: 1
follower:
  - id: 1
    user_id: 1
    follower_id: 2
```

When

The following test is executed: 🍇

```
@Test
@DataSet(value = "datasets/yml/users.yml", useSequenceFiltering =
true)
public void shouldSeedUserDataSet() {
    User user = (User)
EntityManagerProvider.em().createQuery("select u from User u join fetch
u.tweets join fetch u.followers where u.id = 1").getSingleResult();
    assertNotNull(user);
    assertEquals(1, user.getId());
    assertNotNull(user.getTweets().hasSize(1));
    Tweet tweet = user.getTweets().get(0);
    assertNotNull(tweet);
    Calendar date = tweet.getDate();
    Calendar now = Calendar.getInstance();

    assertEquals(date.get(Calendar.DAY_OF_MONTH), now.get(Calendar.
DAY_OF_MONTH));
}
```



Source code of the above example can be [found here](#).

*Then*

The database should be seeded with the dataset content before test execution 🍇

# Chapter 3. Manage database with Database Rider CDI

In order to manage database state in **CDI** based tests  
As a developer  
I want to use DBUnit in a CDI test environment.

DBUnit CDI integration is done through a [CDI interceptor](#) which reads `@DataSet` to prepare database for CDI based tests.

CDI must be enabled in your test, see the following example:



```
@RunWith(CdiTestRunner.class) ①
@DBUnitInterceptor ②
public class DBUnitCDITest {

}
```

① [CdiTestRunner](#) is provided by [Apache Deltaspike](#) but you should be able to use other CDI test runners.

② Needed to activate DBUnit interceptor

## Dependencies

To use this module just add the following maven dependency:

```
<dependency>
  <groupId>com.github.database-rider</groupId>
  <artifactId>rider-cdi</artifactId>
  <version>1.0.0-RC3-SNAPSHOT</version>
  <scope>test</scope>
</dependency>
```

## 3.1. Seed database using yml dataset

*Given*

DBUnit interceptor is enabled in your test beans.xml: 🍻

*src/test/resources/META-INF/beans.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
  <interceptors>

  <class>com.github.database.rider.cdi.DBUnitInterceptorImpl</class>
  </interceptors>
</beans>
```



Your test itself must be a CDI bean to be intercepted. if you're using [Deltaspike test control](#) just enable the following property in *test/resources/META-INF/apache-deltaspike.properties*:

```
deltaspike.testcontrol.use_test_class_as_cdi_bean=true
```

*And*



The following dataset 🍷

*src/test/resources/dataset/yml/users.yml*

```
user:
  - id: 1
    name: "@realpestano"
  - id: 2
    name: "@dbunit"
tweet:
  - id: abcdef12345
    content: "dbunit rules!"
    user_id: 1
  - id: abcdef12233
    content: "dbunit rules!"
    user_id: 2
  - id: abcdef1343
    content: "CDI for the win!"
    user_id: 2
follower:
  - id: 1
    user_id: 1
    follower_id: 2
```

*When*

The following test is executed: 🍇

```
@Test
@DataSet("yaml/users.yaml")
public void shouldSeedUserDataSetUsingCdiInterceptor() {
    List<User> users = em.createQuery("select u from User u order
by u.id asc").getResultList();
    User user1 = new User(1);
    User user2 = new User(2);
    Tweet tweetUser1 = new Tweet();
    tweetUser1.setId("abcdef12345");
    assertThat(users).isNotNull().hasSize(2).contains(user1,
user2);
    List<Tweet> tweetsUser1 = users.get(0).getTweets();

    assertThat(tweetsUser1).isNotNull().hasSize(1).contains(tweetUser1);
}
```



Source code of the above example can be [found here](#).

*Then*

The database should be seeded with the dataset content before test execution 🍇

# Chapter 4. Manage database with Database Rider Cucumber

In order to manage database state in **BDD** tests  
As a BDD developer  
I want to use DBUnit along side my BDD tests.

DBUnit enters the BDD world through a dedicated JUnit runner which is based on [Cucumber](#) and [Apache DeltaSpike](#).

This runner just starts CDI within your BDD tests so you just have to use [Database Rider CDI interceptor](#) on Cucumber steps, here is the so called Cucumber CDI runner declaration:

```
package com.github.database.rider.core.bdd;

import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
import org.junit.runner.RunWith;

/**
 * Created by rmpestano on 4/17/16.
 */
@RunWith(Cucumber.class)
@CucumberOptions(features = {
    "src/test/resources/features/seeding/seeding-database.feature",
    "src/test/resources/features/cdi/cdi-seed-database.feature",
    "src/test/resources/features/cucumber/cucumber-seed-database.feature",
    "src/test/resources/features/junit5/junit5-seed-database.feature",
    "src/test/resources/features/general/dataset-replacements.feature",
    "src/test/resources/features/general/expected-dataset.feature"
},
    plugin = "json:target/dbunit-rules.json")
public class DatabaseRiderBdd {
}
```



As cucumber doesn't work with JUnit Rules, see [this issue](#), you won't be able to use Cucumber runner with *Database Rider Core* because its based on JUnit rules, but you can use DataSetExecutor in [@Before](#), see [example here](#).

## Dependencies

Here is a set of maven dependencies needed by Database Rider Cucumber:



Most of the dependencies, except CDI container implementation, are brought by Database Rider Cucumber module transitively.

```
<dependency>
  <groupId>com.github.database-rider</groupId>
  <artifactId>rider-cucumber</artifactId>
  <version>1.0.0-RC3-SNAPSHOT</version>
  <scope>test</scope>
</dependency>
```

### *Cucumber dependencies*

```
<dependency> ①
  <groupId>info.cukes</groupId>
  <artifactId>cucumber-junit</artifactId>
  <version>1.2.4</version>
  <scope>test</scope>
</dependency>
<dependency> ①
  <groupId>info.cukes</groupId>
  <artifactId>cucumber-java</artifactId>
  <version>1.2.4</version>
  <scope>test</scope>
</dependency>
```

① You don't need to declare because it comes with Database Rider Cucumber module dependency.

```
<dependency> ①
  <groupId>org.apache.deltaspike.modules</groupId>
  <artifactId>deltaspike-test-control-module-api</artifactId>
  <version>${ds.version}</version>
  <scope>test</scope>
</dependency>

<dependency> ①
  <groupId>org.apache.deltaspike.core</groupId>
  <artifactId>deltaspike-core-impl</artifactId>
  <version>${ds.version}</version>
  <scope>test</scope>
</dependency>

<dependency> ①
  <groupId>org.apache.deltaspike.modules</groupId>
  <artifactId>deltaspike-test-control-module-impl</artifactId>
  <version>${ds.version}</version>
  <scope>test</scope>
</dependency>

<dependency> ②
  <groupId>org.apache.deltaspike.cdictrl</groupId>
  <artifactId>deltaspike-cdictrl-owb</artifactId>
  <version>${ds.version}</version>
  <scope>test</scope>
</dependency>

<dependency> ②
  <groupId>org.apache.openwebbeans</groupId>
  <artifactId>openwebbeans-impl</artifactId>
  <version>1.6.2</version>
  <scope>test</scope>
</dependency>
```

① Also comes with DBUnit Rules Cucumber.

② You can use CDI implementation of your choice.

## 4.1. Seed database using Database Rider in Cucumber tests

Given

The following feature 🍌

Feature: Contacts test

As a user of contacts repository

I want to crud contacts

So that I can expose contacts service

Scenario Outline: search contacts

Given we have a list of contacts

When we search contacts by name "<name>"

Then we should find <result> contacts

Examples: examples1

name	result
delta	1
sp	2
querydsl	1
abcd	0

Scenario: delete a contact

Given we have a list of contacts

When we delete contact by id 1

Then we should not find contact 1

*And*

The following dataset 🍷

```
contact:
- id: 1
  name: "deltaspikes"
  email: "users@deltaspikes.apache.org"
  company_id: 1
- id: 2
  name: "querydsl"
  email: "info@mysema.com"
  company_id: 2
- id: 3
  name: "Spring"
  email: "spring@pivotal.io"
  company_id: 3

company:
- id: 1
  name: "Apache"
- id: 2
  name: "Mysema"
- id: 3
  name: "Pivotal"
- id: 4
  name: "Google"
```

*And*

The following Cucumber test 🍌

```
package com.github.database.rider.examples.cucumber;

import com.github.database.rider.cucumber.CdiCucumberTestRunner;
import cucumber.api.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(CdiCucumberTestRunner.class)
@CucumberOptions(
    features = {"src/test/resources/features/contacts.feature"},
    plugin = {"json:target/cucumber.json"}
    //glue = "com.github.dbunit.rules.examples.glues"
)
public class ContactFeature {
}
```

*When*

The following cucumber steps are executed 🍌

```
package com.github.database.rider.examples.cucumber; ①

import com.github.database.rider.core.api.dataset.DataSet;
import com.github.database.rider.cdi.api.DBUnitInterceptor;
import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;
import org.example.jpdomain.Contact;
import org.example.jpdomain.Contact_;
import org.example.service.deltaspike.ContactRepository;

import javax.inject.Inject;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNull;

@DBUnitInterceptor
public class ContactSteps {

    @Inject
    ContactRepository contactRepository; ①

    Long count;
```



```

    @When("^we search contacts by name \"([^\"]*)\"$")
    public void we_search_contacts_by_name_(String name) throws
    Throwable {
        Contact contact = new Contact();
        contact.setName(name);
        count = contactRepository.countLike(contact, Contact_.name);
    }

    @Then("^we should find (\\d+) contacts$")
    public void we_should_find_result_contacts(Long result) throws
    Throwable {
        assertEquals(result, count);
    }

    @Given("^we have a list of contacts$")
    @DataSet("datasets/contacts.yml") ②
    public void given() {
        assertEquals(contactRepository.count(), new Long(3));
    }

    @When("^we delete contact by id (\\d+)$")
    public void we_delete_contact_by_id(long id) throws Throwable {
        contactRepository.remove(contactRepository.findBy(id));
    }

    @Then("^we should not find contact (\\d+)$")
    public void we_should_not_find_contacts_in_database(long id) throws
    Throwable {
        assertNull(contactRepository.findBy(id));
    }
}

```

① As the Cucumber cdi runner enables CDI, you can use injection into your Cucumber steps.

② Dataset is prepared before step execution by `@DBUnitInterceptor`.



Source code for the example above can be [found here](#).

*Then*

The database should be seeded with the dataset content before step execution 🍷

# Chapter 5. Manage database with Database Rider and JUnit 5

In order to manage database state in [JUnit 5](#) integration tests  
As a developer  
I want to use DBUnit along side my JUnit 5 tests.

DBUnit is enabled in JUnit 5 tests through an [extension](#) named **DBUnitExtension**.

## Dependencies

To use the extension just add the following maven dependency:

```
<dependency>
  <groupId>com.github.dbunit-rules</groupId>
  <artifactId>junit5</artifactId>
  <version>1.0.0-RC3-SNAPSHOT</version>
  <scope>test</scope>
</dependency>
```

## 5.1. Seed database using Database Rider in JUnit5 tests

*Given*

The following dataset 🍌

*src/test/resources/dataset/users.yml*

```
user:
  - id: 1
    name: "@realpestano"
  - id: 2
    name: "@dbunit"
```

*When*

The following junit5 test is executed 🍻

```
@ExtendWith(DBUnitExtension.class) ①
@RunWith(JUnitPlatform.class) ②
@DataSet(cleanBefore = true)
public class DBUnitJUnit5It {

    private ConnectionHolder connectionHolder = () -> ③
        EntityManagerProvider.instance("junit5-
        pu").clear().connection();④

    @Test
    @DataSet(value = "usersWithTweet.yml")
    public void shouldListUsers() {
        List<User> users =
        EntityManagerProvider.em().createQuery("select u from User
        u").getResultList();
        assertThat(users).isNotNull().isNotEmpty().hasSize(2);
    }
}
```

① Enables DBUnit;

② JUnit 5 runner;

③ As JUnit5 requires **Java8** you can use lambdas in your tests;

④ DBUnitExtension will get connection by reflection so just declare a field or a method with **ConnectionHolder** as return type.



Source code of the above example can be [found here](#).

*Then*

The database should be seeded with the dataset content before test execution 🍻

# Chapter 6. Dynamic data using scritable datasets

In order to have dynamic data in datasets  
As a developer  
I want to use scripts in DBUnit datasets.

Scritable datasets are backed by JSR 223. [2: Scripting for the Java Platform, for more information access the official [docs here](#)].

Complete source code of examples below can be [found here](#).

## 6.1. Seed database with groovy script in dataset

*Given*

Groovy script engine is on test classpath 🍻

```
<dependency>
  <groupId>org.codehaus.groovy</groupId>
  <artifactId>groovy-all</artifactId>
  <version>2.4.6</version>
  <scope>test</scope>
</dependency>
```

*And*

The following dataset 🍻

```
tweet:
- id: "1"
  content: "dbunit rules!"
  date: "groovy:new Date()" ①
  user_id: 1
```

① Groovy scripting is enabled by **groovy:** string.

When

The following test is executed: 🍇

```
@Test
@DataSet(value = "datasets/yml/groovy-with-date-
replacements.yml",cleanBefore = true, disableConstraints = true,
executorId = "rules-it")
public void shouldReplaceDateUsingGroovyInDataset() {
    Tweet tweet = (Tweet) emProvider.em().createQuery("select t from
    Tweet t where t.id = '1'").getSingleResult();
    assertThat(tweet).isNotNull();

    assertThat(tweet.getDate().get(Calendar.DAY_OF_MONTH)).isEqualTo(now.ge
    t(Calendar.DAY_OF_MONTH));

    assertThat(tweet.getDate().get(Calendar.HOUR_OF_DAY)).isEqualTo(now.get
    (Calendar.HOUR_OF_DAY));
}
```



Source code of the above example can be [found here](#).

Then

Dataset script should be interpreted while seeding the database 🍇

## 6.2. Seed database with javascript in dataset



Javascript engine comes within JDK so no additional classpath dependency is necessary.

*Given*

The following dataset 🍌

```
tweet:
  - id: "1"
    content: "dbunit rules!"
    likes: "js:(5+5)*10/2" ①
    user_id: 1
```

① Javascript scripting is enabled by `js:` string.

*When*

The following test is executed: 🍌

```
@Test
@DataSet(value = "datasets/yml/js-with-calc-
replacements.yml",cleanBefore = true ,disableConstraints = true,
executorId = "rules-it")
public void shouldReplaceLikesUsingJavaScriptInDataset() {
    Tweet tweet = (Tweet) emProvider.em().createQuery("select t from
Tweet t where t.id = '1'").getSingleResult();
    assertNotNull(tweet);
    assertEquals(50,tweet.getLikes());
}
```



Source code of the above example can be [found here](#).

*Then*

Dataset script should be interpreted while seeding the database 🍌

# Chapter 7. Database assertion using expected datasets

In order to verify database state after test execution  
As a developer  
I want to assert database state with datasets.

Complete source code of examples below can be [found here](#).

## 7.1. Database assertion with yaml dataset

*Given*

The following dataset 🍌

*expectedUsers.yml*

```
user:
  - id: 1
    name: "expected user1"
  - id: 2
    name: "expected user2"
```

*When*

The following test is executed: 🍌

```
@RunWith(JUnit4.class)
@DBUnit(cacheConnection = true)
public class ExpectedDataSetIt {

    @Rule
    public EntityManagerProvider emProvider =
        EntityManagerProvider.instance("rules-it");

    @Rule
    public DBUnitRule dbUnitRule =
        DBUnitRule.instance(emProvider.connection());

    @Test
    @DataSet(cleanBefore = true)①
    @ExpectedDataSet(value = "yaml/expectedUsers.yaml", ignoreCols = "id")
    public void shouldMatchExpectedDataSet() {
        EntityManagerProvider instance =
            EntityManagerProvider.newInstance("rules-it");
        User u = new User();
        u.setName("expected user1");
        User u2 = new User();
        u2.setName("expected user2");
        instance.tx().begin();
        instance.em().persist(u);
        instance.em().persist(u2);
        instance.tx().commit();
    }
}
```

① Clear database before to avoid conflict with other tests.

*Then*

Test must pass because database state is as in expected dataset. 🍌

## 7.2. Database assertion with regular expression in expected dataset



*Given*

The following dataset 🍌

*expectedUsersRegex.yml*

```
user:
  - id: "regex:\\d+"
    name: regex:^expected user.* #expected user1
  - id: "regex:\\d+"
    name: regex:.*user2$ #expected user2
```

*When*

The following test is executed: 🍌

```
@Test
@DataSet(cleanBefore = true)
@ExpectedDataSet(value = "yaml/expectedUsersRegex.yml")
public void shouldMatchExpectedDataSetUsingRegex() {
    User u = new User();
    u.setName("expected user1");
    User u2 = new User();
    u2.setName("expected user2");
    EntityManagerProvider.tx().begin();
    EntityManagerProvider.em().persist(u);
    EntityManagerProvider.em().persist(u2);
    EntityManagerProvider.tx().commit();
}
```

*Then*

Test must pass because database state is as in expected dataset. 🍌

## 7.3. Database assertion with seeding before test execution

*Given*

The following dataset 🍌

*user.yml*

```
user:
  - id: 1
    name: "@realpestano"
  - id: 2
    name: "@dbunit"
```

*And*

The following dataset 🍌

*expectedUser.yml*

```
user:
  - id: 2
    name: "@dbunit"
```

*When*

The following test is executed: 🍌

```
@Test
@DataSet(value = "yaml/user.yml", disableConstraints = true)
@ExpectedDataSet(value = "yaml/expectedUser.yml", ignoreCols = "id")
public void shouldMatchExpectedDataSetAfterSeedingDataBase() {
    tx().begin();
    em().remove(EntityManagerProvider.em().find(User.class, 1L));
    tx().commit();
}
```

*Then*

Test must pass because database state is as in expected dataset. 🍌

## 7.4. Failing database assertion

*Given*

The following dataset 🍌

*expectedUsers.yml*

```
user:
  - id: 1
    name: "expected user1"
  - id: 2
    name: "expected user2"
```

*When*

The following test is executed: 🍌

```
@Test
@ExpectedDataSet(value = "yml/expectedUsers.yml", ignoreCols = "id")
public void shouldNotMatchExpectedDataSet() {
    User u = new User();
    u.setName("non expected user1");
    User u2 = new User();
    u2.setName("non expected user2");
    EntityManagerProvider.tx().begin();
    EntityManagerProvider.em().persist(u);
    EntityManagerProvider.em().persist(u2);
    EntityManagerProvider.tx().commit();
}
```

*Then*

Test must fail with following error: 🍷



```
junit.framework.ComparisonFailure: value (table=USER, row=0,
col=name) expected:<[]expected user1> but was:<[non ]expected
user1>                                     at
org.dbunit.assertion.JUnitFailureFactory.createFailure(JUnitFailur
eFactory.java:39)                           at
org.dbunit.assertion.DefaultFailureHandler.createFailure(Default
FailureHandler.java:97)                       at
org.dbunit.assertion.DefaultFailureHandler.handle(DefaultFailure
Handler.java:223) at ...
```

## 7.5. Database assertion using automatic transaction

## Given

The following dataset 🍌

*expectedUsersRegex.yml*

```
user:
  - id: "regex:\\d+"
    name: regex:^expected user.* #expected user1
  - id: "regex:\\d+"
    name: regex:.*user2$ #expected user2
```

## When

The following test is executed: 🍌

```
@Test
@DataSet(cleanBefore = true, transactional = true, executorId =
"TransactionIt")
@ExpectedDataSet(value = "yml/expectedUsersRegex.yml")
@DBUnit(cacheConnection = true)
public void shouldManageTransactionAutomatically() {
    User u = new User();
    u.setName("expected user1");
    User u2 = new User();
    u2.setName("expected user2");
    EntityManagerProvider.em().persist(u);
    EntityManagerProvider.em().persist(u2);
}
```



**Transactional** attribute will make Database Rider start a transaction before test and commit the transaction **after** test execution but **before** expected dataset comparison.

## Then

Test must pass because inserted users are committed to database and database state matches expected dataset. 🍌