

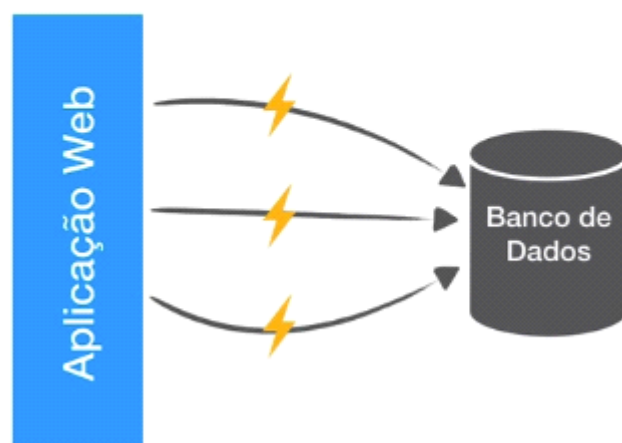
7.1. Entendendo o funcionamento de um pool de conexões

segunda-feira, 20 de fevereiro de 2023 21:50



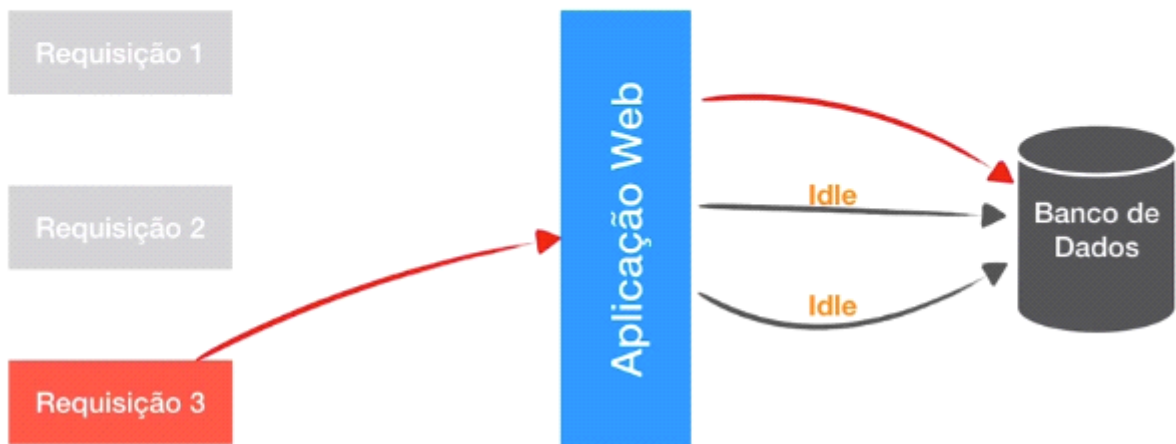
É um componente de soft que mantém um conjunto de conexões com o bd para reutilização na aplicação

Aplicação **com** pool de conexões



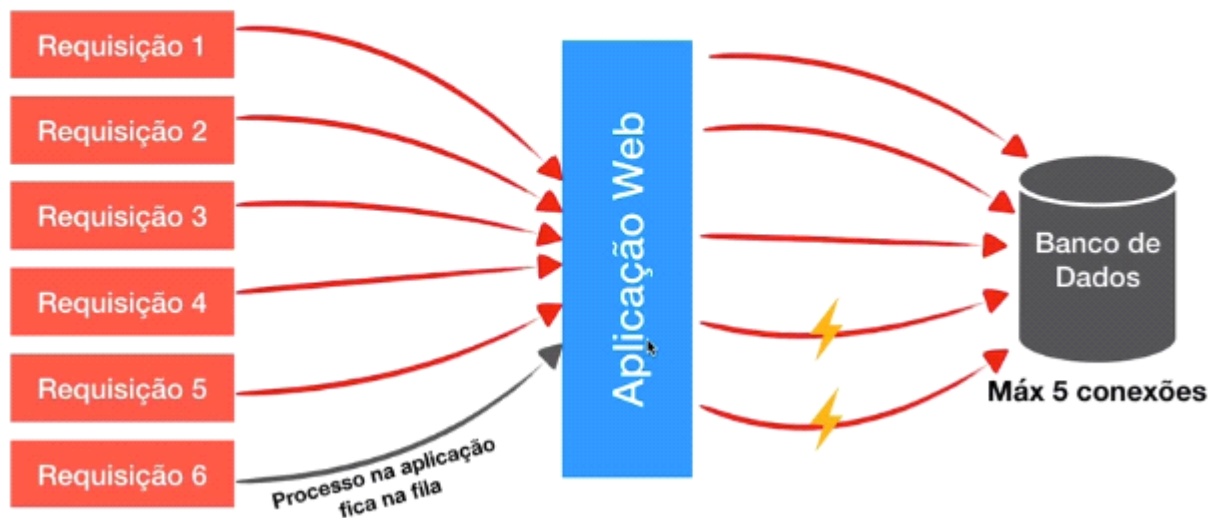
Spring cria o pool de conexões ao iniciar a aplicação. Ao criar tais conexões e por enquanto sem aplicações acessando essas conexões, elas ficam em estado de Idle

Aplicação **com** pool de conexões



Existe um número mínimo de pools iniciais a partir de uma configuração e um número máximo de pools para ser estabelecido.

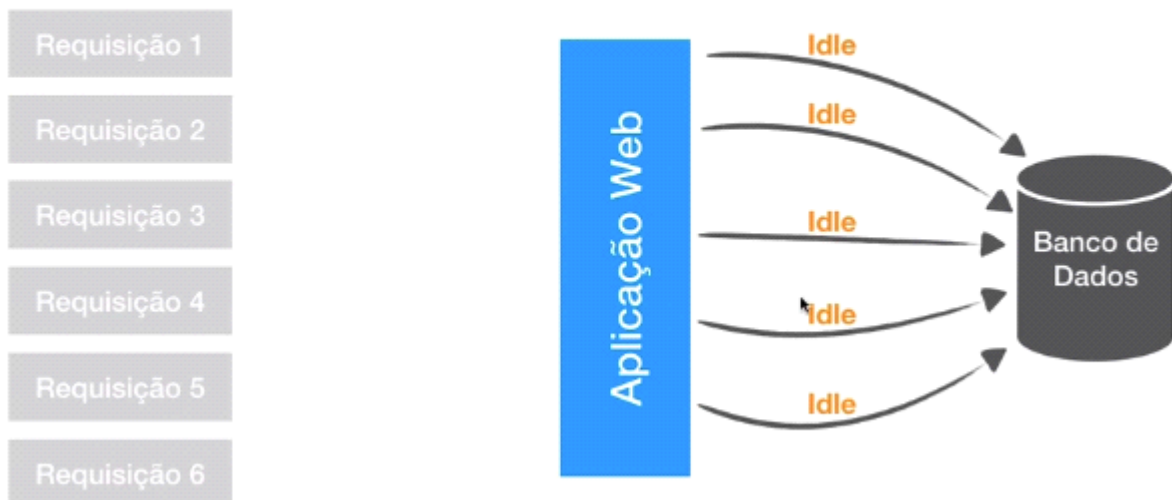
Aplicação **com** pool de conexões



A aplicação primeiramente iniciará o pool de conexões com o número mínimo estabelecido e caso haja necessidade de usar mais conexões por conta de requisições a aplicação, a mesma estabelecerá mais conexões de acordo com o número máximo estabelecido.

Caso haja mais requisições que o limite suportado, os demais processos ficarão em espera até que uma conexão entre em estado de idle e entregue para o processo em espera.

Aplicação **com** pool de conexões

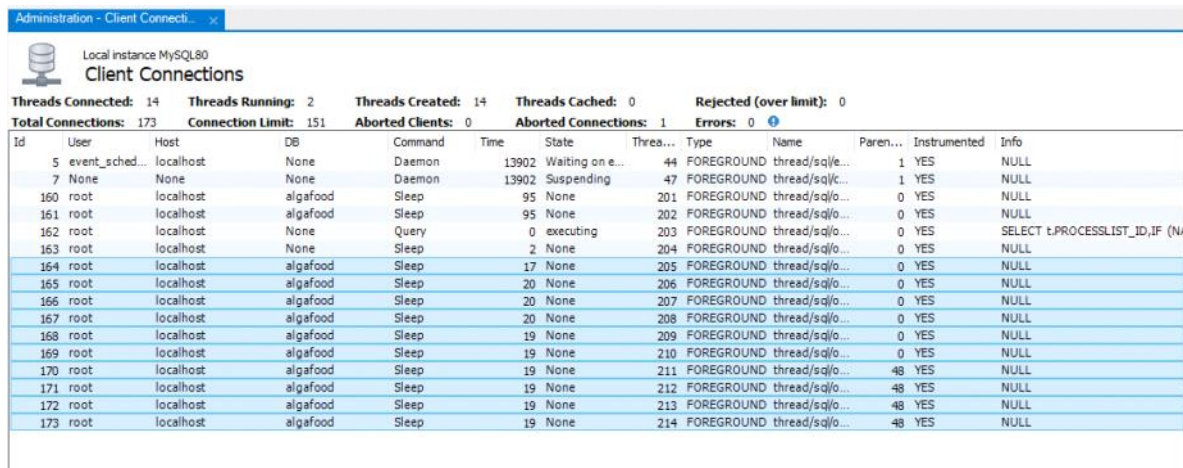


Geralmente, após os processos serem finalizados, a aplicação continuará com as conexões excedentes no pool por um período de tempo determinado por uma configuração.

Reduz o tempo do usuário ou uma requisição a api responder com sucesso os dados, já que otimiza o uso com o banco de dados.

7.2. Conhecendo o Hikari a solução padrão de pool de conexões no Spring Boot

segunda-feira, 20 de fevereiro de 2023 22:12



Administration - Client Connections

Local instance MySQL80

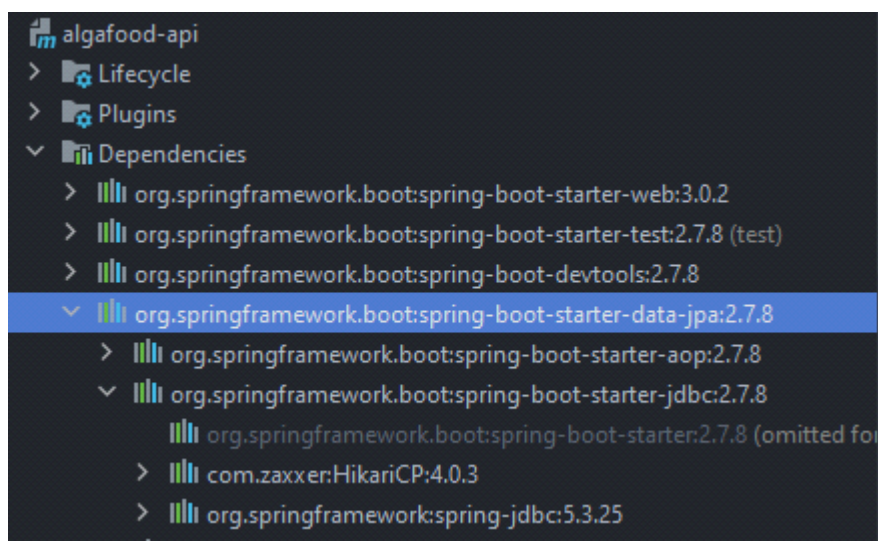
Client Connections

Threads Connected: 14 Threads Running: 2 Threads Created: 14 Threads Cached: 0 Rejected (over limit): 0

Total Connections: 173 Connection Limit: 151 Aborted Clients: 0 Aborted Connections: 1 Errors: 0

Id	User	Host	DB	Command	Time	State	Threa...	Type	Name	Paren...	Instrumented	Info
5	event_sched...	localhost	None	Daemon	13902	Waiting on e...	44	FOREGROUND	thread/sql/e...	1	YES	NULL
7	None	None	None	Daemon	13902	Suspending	47	FOREGROUND	thread/sql/c...	1	YES	NULL
160	root	localhost	algafood	Sleep	95	None	201	FOREGROUND	thread/sql/o...	0	YES	NULL
161	root	localhost	algafood	Sleep	95	None	202	FOREGROUND	thread/sql/o...	0	YES	NULL
162	root	localhost	None	Query	0	executing	203	FOREGROUND	thread/sql/o...	0	YES	SELECT t.PROCESSLIST_ID,IF (NA
163	root	localhost	None	Sleep	2	None	204	FOREGROUND	thread/sql/o...	0	YES	NULL
164	root	localhost	algafood	Sleep	17	None	205	FOREGROUND	thread/sql/o...	0	YES	NULL
165	root	localhost	algafood	Sleep	20	None	206	FOREGROUND	thread/sql/o...	0	YES	NULL
166	root	localhost	algafood	Sleep	20	None	207	FOREGROUND	thread/sql/o...	0	YES	NULL
167	root	localhost	algafood	Sleep	20	None	208	FOREGROUND	thread/sql/o...	0	YES	NULL
168	root	localhost	algafood	Sleep	19	None	209	FOREGROUND	thread/sql/o...	0	YES	NULL
169	root	localhost	algafood	Sleep	19	None	210	FOREGROUND	thread/sql/o...	0	YES	NULL
170	root	localhost	algafood	Sleep	19	None	211	FOREGROUND	thread/sql/o...	48	YES	NULL
171	root	localhost	algafood	Sleep	19	None	212	FOREGROUND	thread/sql/o...	48	YES	NULL
172	root	localhost	algafood	Sleep	19	None	213	FOREGROUND	thread/sql/o...	48	YES	NULL
173	root	localhost	algafood	Sleep	19	None	214	FOREGROUND	thread/sql/o...	48	YES	NULL

Ao iniciar a aplicação Spring, o próprio Spring Boot adiciona dependências que configuram o pool de conexões ao levantar a aplicação.



A dependência HikariCP que configura o pool de conexões

7.3. Configurando o pool de conexões do Hikari

segunda-feira, 20 de fevereiro de 2023 22:45

```
spring.datasource.hikari.maximum-pool-size=5  
spring.datasource.hikari.minimum-idle=3  
spring.datasource.hikari.idle-timeout=10000
```

a definição máxima de pool são 5, além do mínimo de conexões iniciais 3, e idle-timeout para eliminar as conexões excedentes pelo tempo definido em milissegundos

7.4. Schema generation em produção não é uma boa prática

segunda-feira, 20 de fevereiro de 2023 22:51

O Schema Generation é a ferramenta do Hibernate para criar tabelas no banco de dados a partir do mapeamento de entidades, podemos configurar a forma da criação dessas tabelas no `application.properties`

```
#gerando as tabelas automaticamente script de criacao
spring.jpa.generate-ddl=true

#configuracao especifica do hibernate p/ qual forma o ddl vai ser executado
spring.jpa.hibernate.ddl-auto=create
```

o valor `create` deleta tudo o que tiver no banco de dados na hora da inicialização da aplicação e recria as tabelas e dados de um arquivo (caso houver arquivo) com nome `import.sql`.

Não é uma boa opção utilizá-lo quando já houver registros no banco, pois o Schema Generation apaga todos os dados da tabela. É recomendável usar em projetos pequenos, de prototipação e testes.

Mesmo se mudarmos o schema para `update`, haverá problemas para atualização de tabelas e dados, pois pode haver dados e tabelas inconsistentes.

O ideal é utilizar ferramentas de criação de scripts de migrações incrementais

7.5. Flyway ferramenta de versionamento de schemas de banco de dados

terça-feira, 21 de fevereiro de 2023

12:52



Ferramenta de versionamento de schemas de bancos de dados

Como **profissionais** gerenciam as mudanças em schemas do banco de dados



7.6. Adicionando o Flyway no projeto e criando a primeira migração

terça-feira, 21 de fevereiro de 2023 13:11

Para utilizar o Flyway no projeto, é necessário desabilitar o Schema Generation do Hibernate

```
#gerando as tabelas automaticamente script de criacao
#spring.jpa.generate-ddl=true

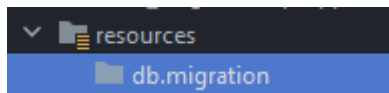
#configuracao especifica do hibernate p/ qual forma o ddl vai ser executado
#spring.jpa.hibernate.ddl-auto=create
```

Dropar todas as tabelas do banco de dados e adicionar a dependência do Flyway no Maven

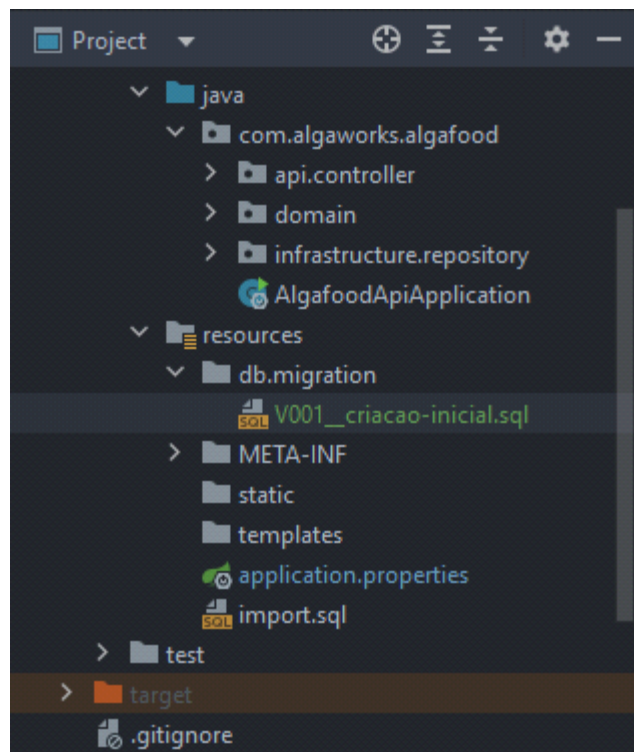
```
<!--https://mvnrepository.com/artifact/org.flywaydb/flyway-core-->
<dependency>
<groupId>org.flywaydb</groupId>
<artifactId>flyway-core</artifactId>
</dependency>

<dependency>
<groupId>org.flywaydb</groupId>
<artifactId>flyway-mysql</artifactId>
</dependency>
```

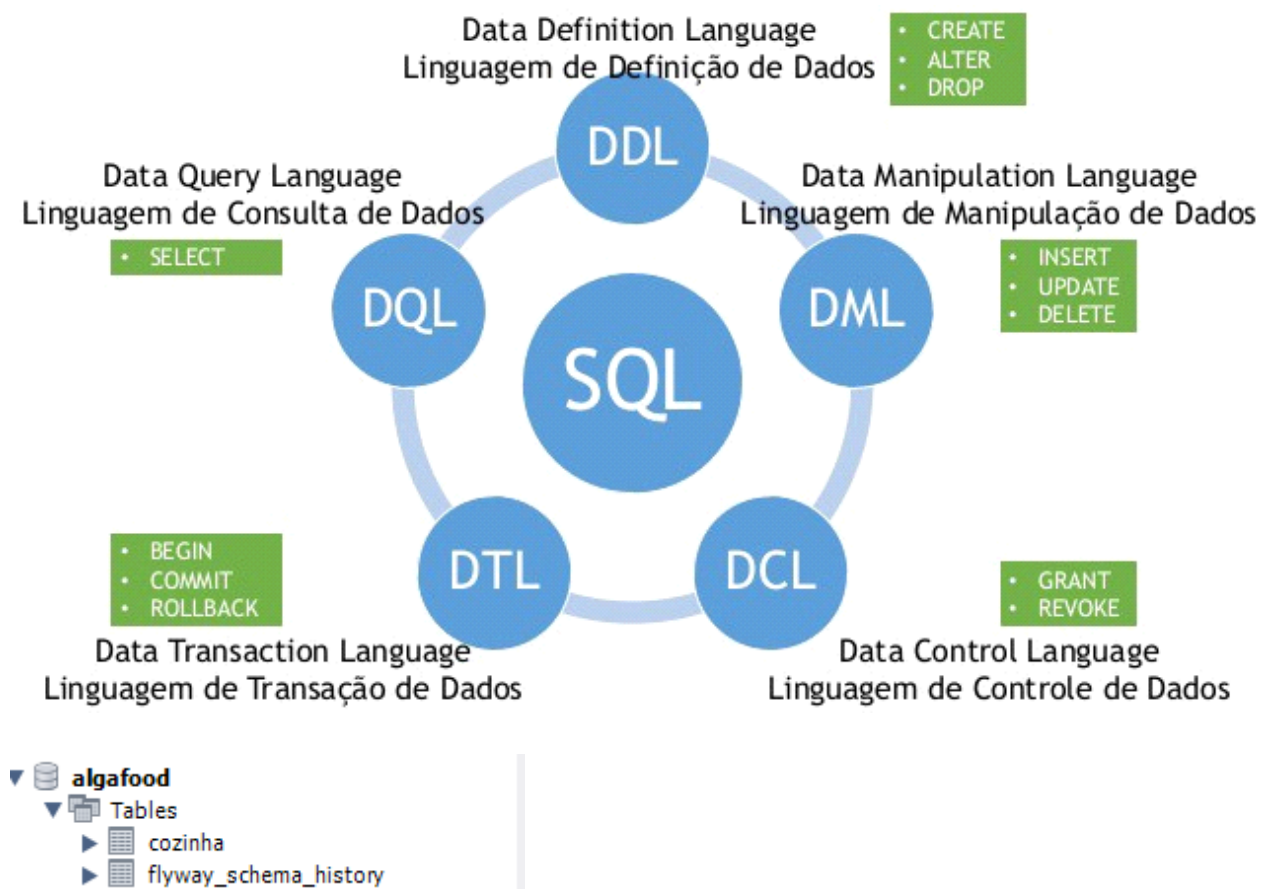
e adicionar a pasta db/migration dentro de resource



Agora podemos criar nossos scripts sql de migração

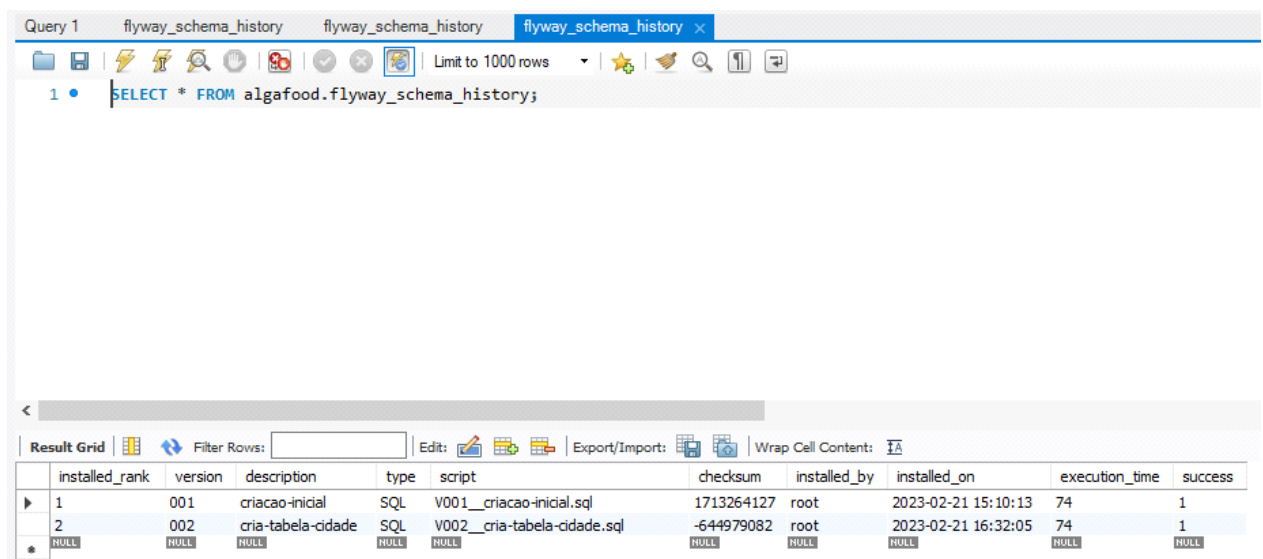
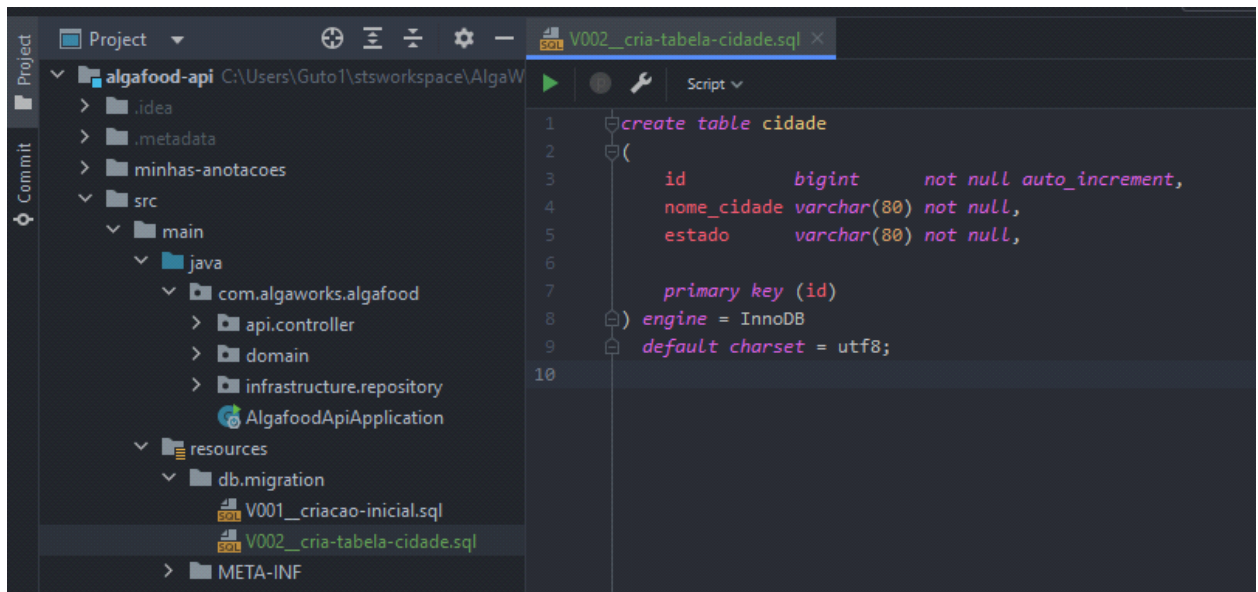


Cada script de migração possui um padrão para checagem de histórico de alteração, começando com V maiúsculo, a contagem de script, podendo ser um valor incremental como usado no script acima, ou contagem de data e hora V20230219184530 - 2022/02/19 18:45:30 ou por versão em ponto (.) ou underline (_) V1.1 V1_8. Seguido de dois underlines (__) para adicionar uma descrição da migração. Dentro do arquivo SQL deve conter os script SQL, não é recomendável adicionar script de manipulação DML, apenas DDL para definição de dados, pois a estrutura das migrações serão utilizadas em produção, os dados serão populados pelos usuários



7.7. Evoluindo o banco de dados com novas migrações

terça-feira, 21 de fevereiro de 2023 16:14



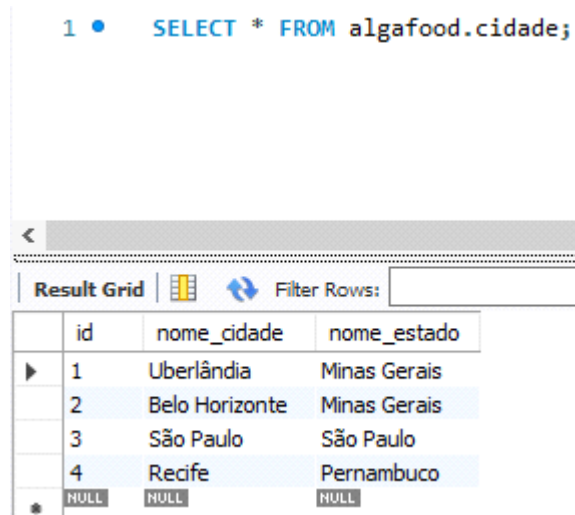
Caso haja erro de inconsistência de tabelas, basta deleta o campo da tentativa, porém, tem que se certificar que é seguro fazer isso

7.8. Criando migrações complexas com remanejamento de dados

terça-feira, 21 de fevereiro de 2023 16:34

Problema: Uma tabela já populada com colunas que deverão ser adicionadas em uma tabela separa.

```
1 • SELECT * FROM algafood.cidade;
```



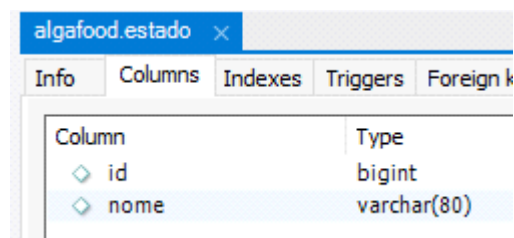
	id	nome_cidade	nome_estado
▶	1	Uberlândia	Minas Gerais
	2	Belo Horizonte	Minas Gerais
	3	São Paulo	São Paulo
	4	Recife	Pernambuco
*	NULL	NULL	NULL

Temos a coluna nome_estado da tabela cidade que deverá migrar para outra tabela, e os campos da coluna nome_estado deverão ser substituídos por chaves estrangeiras.

1º Criando a tabela de estado

```
create table estado (  
id bigint not null auto_increment,  
nome varchar(80) not null,
```

```
primary key (id)  
) engine = InnoDB default charset=utf8;
```



Column	Type
id	bigint
nome	varchar(80)

2º Adicionando registros na tabela estado na coluna nome a partir dos campos da coluna nome_estado da tabela cidade

```
insert into estado (nome)  
select distinct nome_estado from cidade;
```



Result Grid		
	id	nome
▶	1	Minas Gerais
	2	São Paulo
	3	Pernambuco
✱	NULL	NULL

3º Adicionando coluna estado_id na tabela cidade para referenciar os campos da tabela cidade na tabela estado

```
alter table cidade
add column estado_id bigint not null;
```

```
1 • SELECT * FROM algafood.cidade;
```

<

Result Grid   Filter Rows:

	id	nome_cidade	nome_estado	estado_id
▶	1	Uberlândia	Minas Gerais	0
	2	Belo Horizonte	Minas Gerais	0
	3	São Paulo	São Paulo	0
	4	Recife	Pernambuco	0
⬆	NULL	NULL	NULL	NULL



4º Setando os id's na coluna estado_id da tabela cidade correspondentes ao id de cada estado da tabela estado

```
update cidade c
set c.estado_id =
(select e.id from estado e where e.nome = c.nome_estado);
```

```
1 • SELECT * FROM algafood.cidade;
```

<

.....

Result Grid   Filter Rows:

	id	nome_cidade	nome_estado	estado_id
▶	1	Uberlândia	Minas Gerais	1
	2	Belo Horizonte	Minas Gerais	1
	3	São Paulo	São Paulo	2
	4	Recife	Pernambuco	3
	NULL	NULL	NULL	NULL

4º Agora vamos adicionar a foreign key da tabela cidade

```
alter table cidade
```

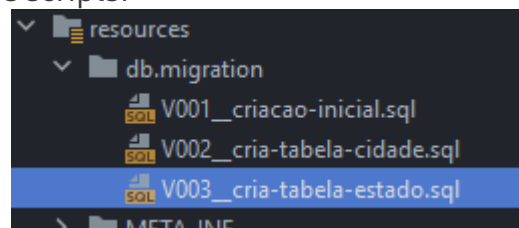
```
add constraint fk_cidade_estado  
foreign key (estado_id)  
references estado(id);
```

5º Podemos deletar a coluna nome_estado e alterar a coluna nome_cidade para apenas nome

```
alter table cidade  
drop column nome_estado;
```

```
alter table cidade  
change nome_cidade nome varchar(80) not null;
```

6º Para adicionar o script de remanejamento de dados para gerencia do Flyway, precisamos deletar as alterações de testes para a criação do Script, o ideal seria ter realizado um backup/dump antes mas como temos um histórico de alterações com scripts na aplicação, podemos deletar toda a base de dados e ao iniciar a aplicação, o Flyway roda todos os scripts.



7.9. Criando migração a partir de DDL gerado por schema generation

quarta-feira, 22 de fevereiro de 2023 10:34

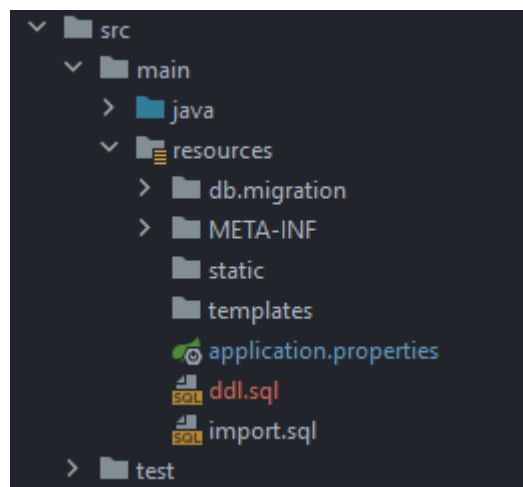
Podemos criar scripts de entidades mapeadas pelo JPA para reutilizar em nossas próprias migrations ou quaisquer outras finalidades.

```
#Os 2 comandos abaixo deverão ser executados apenas 1 vez p/ criação DDL, depois comentados.  
  
#Apenas para criar o script sql a partir das entidades mapeadas  
spring.jpa.properties.javax.persistence.schema-generation.scripts.action=create  
  
#Definindo o caminho do script  
spring.jpa.properties.javax.persistence.schema-generation.scripts.create-target=src/main/resources/ddl.sql
```

#Os 2 comandos abaixo deverão ser executados apenas 1 vez p/ criação DDL, depois comentados.

#Apenas para criar o script sql a partir das entidades mapeadas
`spring.jpa.properties.javax.persistence.schema-generation.scripts.action=create`

#Definindo o caminho do script
`spring.jpa.properties.javax.persistence.schema-generation.scripts.create-target=src/main/resources/ddl.sql`



Depois de criado o script, o comando deve ser comentado pois é necessário somente uma vez.

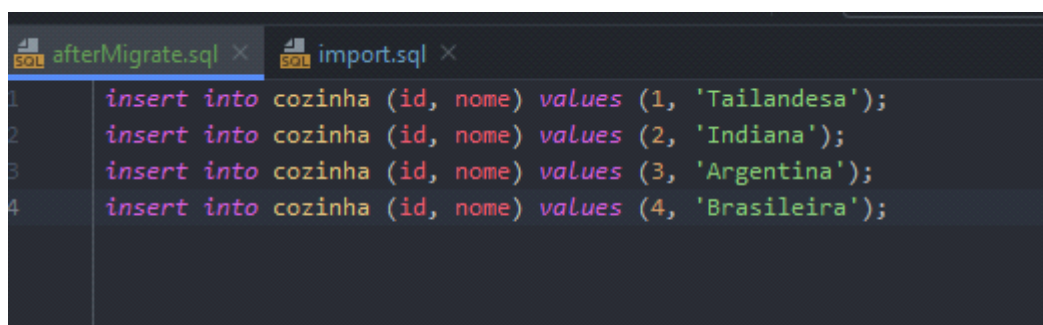
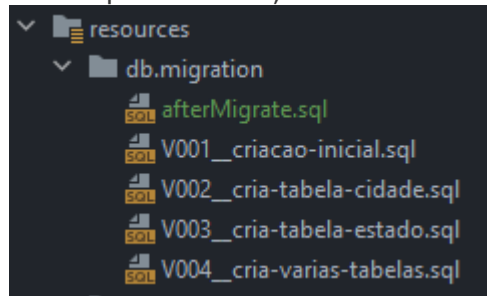

```
application.properties × ddl.sql × V004_cria-varias-tabelas.sql ×
1 # DDL criado pelo Schema-Generation o application.properties e será revisado e usado na migration V004
2 create table cidade (
3     id bigint not null auto_increment,
4     nome varchar(255) not null,
5     estado_id bigint not null,
6     primary key (id)
7 ) engine=InnoDB;
8
9 create table cozinha (
10     id bigint not null auto_increment,
11     nome varchar(30) not null,
12     primary key (id)
13 ) engine=InnoDB;
14
15 create table estado (
16     id bigint not null auto_increment,
17     nome varchar(255) not null,
18     primary key (id)
19 ) engine=InnoDB;
20
21 create table forma_pagamento (
22     id bigint not null auto_increment,
23     descricao varchar(255) not null,
24     primary key (id)
25 ) engine=InnoDB;
26
27 create table grupo (
28     id bigint not null auto_increment,
29     nome varchar(255) not null,
```


7.10. Adicionando dados de testes com callback do Flyway

quarta-feira, 22 de fevereiro de 2023 11:23

Até o momento antes do uso do **Flyway**, usávamos o `import.sql` do schema generation do Hibernate para inserir dados de testes na nossa aplicação, já que o uso do Flyway para inserir dados não é recomendado.

Podemos utilizar o Flyway para inserir dados de forma mais controlada, utilizando o **AfterMigrate**, que é a fase após a execução de todas as migrations do FlyWay.



```
: Schema 'algafood' is up to date. No migration necessary.  
: Executing SQL callback: afterMigrate -
```

Mas ao executar novamente, temos um erro, pois os registros foram adicionados no banco mas não foram deletados.

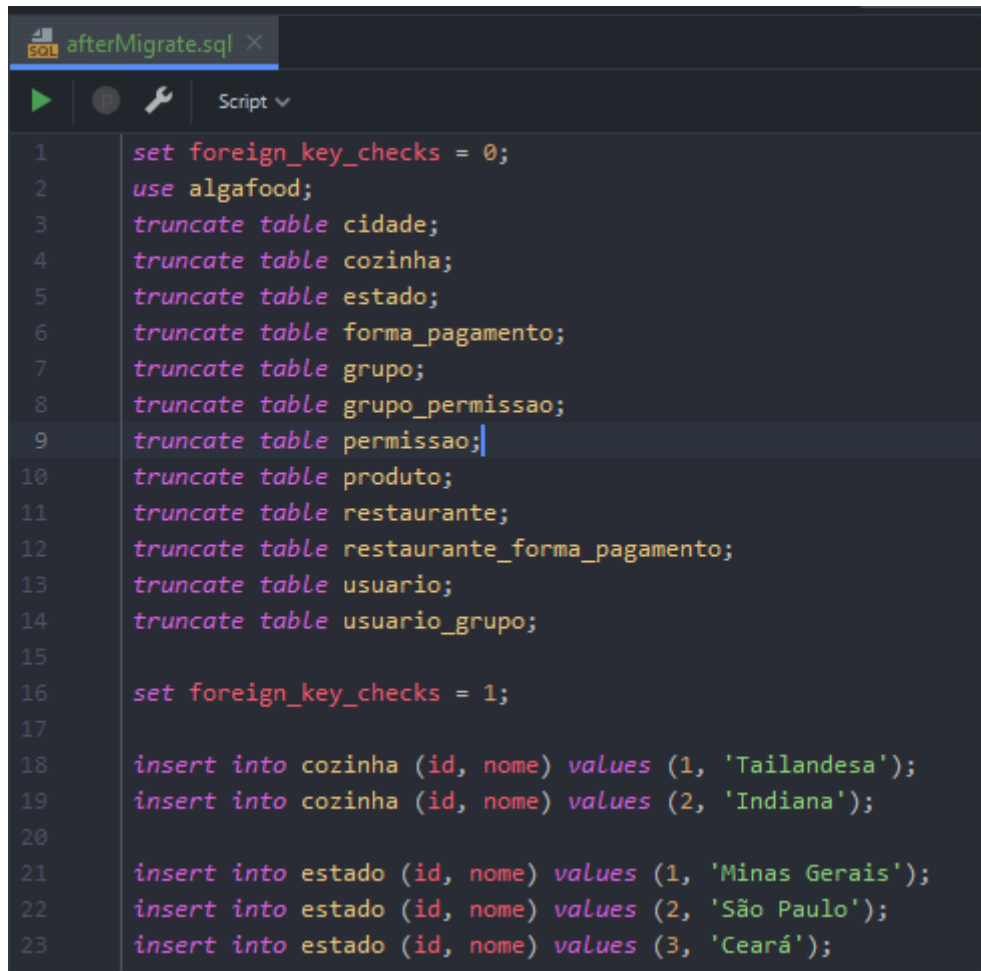
```
Caused by: org.flywaydb.core.internal.sqlscript.FlywaySqlScriptException:  
-----  
SQL State : 23000  
Error Code : 1062  
Message : Duplicate entry '1' for key 'cozinha.PRIMARY'  
Location : db/migration/afterMigrate.sql (C:\Users\Guto1\stsworkspace\Al  
Line : 1  
Statement : insert into cozinha (id, nome) values (1, 'Tailandesa')
```

Podemos utilizar também a palavra reservada `ignore` no comando SQL para executar ou ignorar caso haja falha em uma das linhas.

```
insert ignore into cozinha (id, nome) values (1, 'Tailandesa');  
insert ignore into cozinha (id, nome) values (2, 'Indiana');  
insert ignore into cozinha (id, nome) values (3, 'Argentina');  
insert ignore into cozinha (id, nome) values (4, 'Brasileira');
```

Contudo, os dados inseridos ficarão permanentemente na tabela mesmo depois da aplicação fechar (diferente quando usado schema-generation que dropa as tabelas ao finalizar a aplicação)

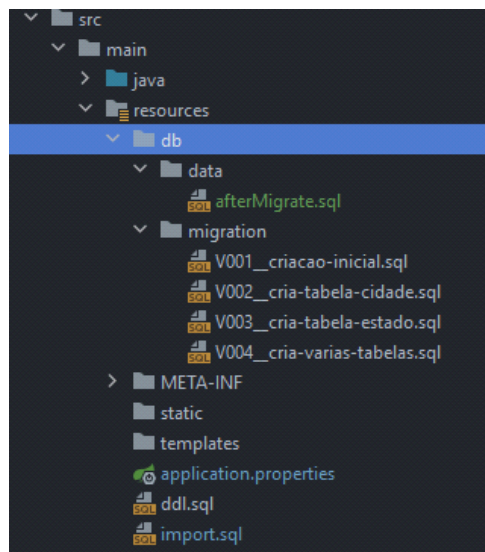
A opção mais viável é criar comandos no afterMigrate para deletar os campos da tabela e inserir dados novamente, como abaixo:



```
1  set foreign_key_checks = 0;
2  use algafood;
3  truncate table cidade;
4  truncate table cozinha;
5  truncate table estado;
6  truncate table forma_pagamento;
7  truncate table grupo;
8  truncate table grupo_permissao;
9  truncate table permissao;
10 truncate table produto;
11 truncate table restaurante;
12 truncate table restaurante_forma_pagamento;
13 truncate table usuario;
14 truncate table usuario_grupo;
15
16 set foreign_key_checks = 1;
17
18 insert into cozinha (id, nome) values (1, 'Tailandesa');
19 insert into cozinha (id, nome) values (2, 'Indiana');
20
21 insert into estado (id, nome) values (1, 'Minas Gerais');
22 insert into estado (id, nome) values (2, 'São Paulo');
23 insert into estado (id, nome) values (3, 'Ceará');
```

Agora sempre que iniciamos a aplicação, os comandos serão executados para deletar os dados (truncate para zerar os registros auto_increment) e foreign_key_checks para ativar e desativar checagem de fk na hora de deletar.

Agora podemos separar esses dados de testes em um diretório separados (db/data) a fim de especificar em quais diretórios o flyway executará os scripts



em `db/data` em `application.properties` -> `spring.flyway.locations`

```
#7.10 Adicionando dados de testes com callback do Flyway  
#indicando ao Flyway as migrations e os dados de teste  
spring.flyway.locations=classpath:db/migration,classpath:db/data
```

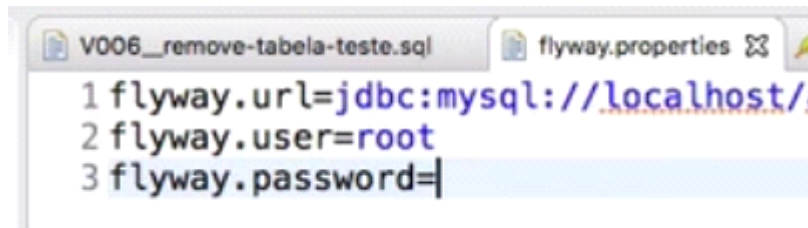
a ideia é separar o `application.properties` de dev e prod, logicamente o perfil de dev conterá o diretório do `afterMigrate`, o de prod não.

7.11. Reparando migrações com erros

quarta-feira, 22 de fevereiro de 2023 19:14

Podemos reparar a tabela de histórico de migrações do banco de dados a partir de um terminal que dê acesso ao path da aplicação através do MAVEN.

Primeiro criamos um arquivo de propriedades do flyway chamado `flyway.properties` para adicionarmos as credenciais de acesso ao banco de dados.



Em seguida podemos acessar o terminal a partir do caminho da aplicação ou classpath da aplicação

```
algafood-api$ ./mvnw flyway:repair -Dflyway.configFiles=src/main/resources/flyway.properties
```

e executar o comando de reparação indicando o caminho do arquivo de propriedade

`./mvnw flyway:repair -Dflyway.configFiles=src/main/resources/flyway.properties` para remover campos da tabela do flyway que não executaram com sucesso

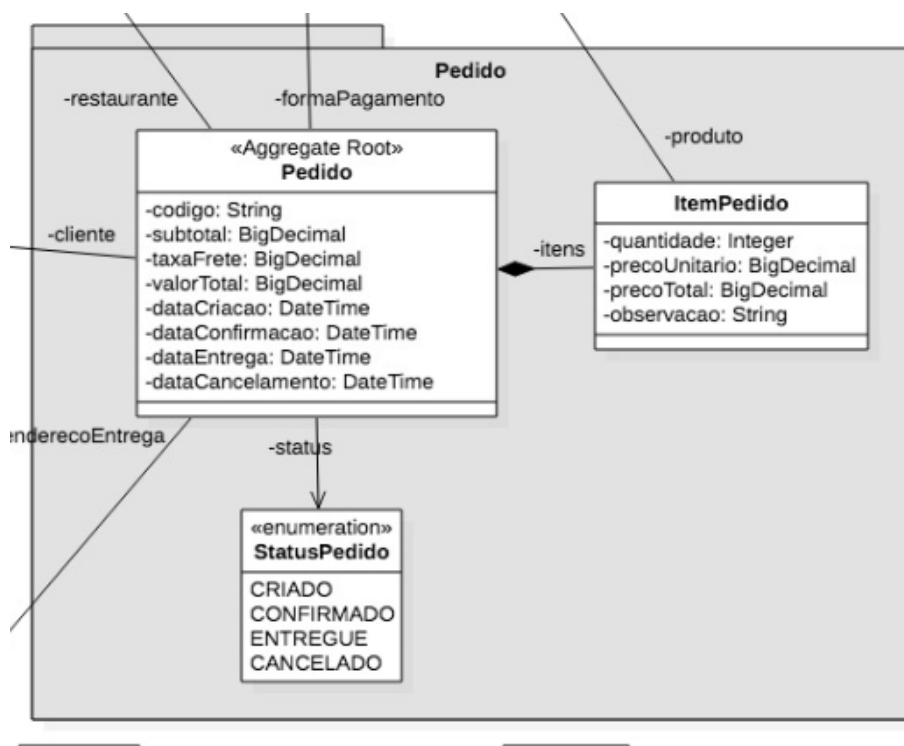
```
Guto1@DESKTOP-EMCTN52 MINGW64 ~/stsworkspace/AlgaWorks/algafood-api (main)
$ ./mvnw flyway:repair -Dflyway.configFiles=src/main/resources/flyway.properties
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.algaworks:algafood-api >-----
[INFO] Building algafood-api 2.1.7.RELEASE
[INFO] -----[ jar ]-----
[WARNING] The artifact mysql:mysql-connector-java:jar:8.0.32 has been relocated
to com.mysql:mysql-connector-j:jar:8.0.32: MySQL Connector/J artifacts moved to
reverse-DNS compliant Maven 2+ coordinates.
[INFO]
```

e agora podemos refazer a migration da forma correta e iniciar a aplicação

7.12. Desafio Criando migrações e mapeando as entidades Pedido e ItemPedido

quinta-feira, 23 de fevereiro de 2023

17:22



Vamos criar a entidade **Pedido** e adicionar as propriedades e relacionamentos com as entidades **FormaPagamento**, **Restaurante**, **Usuario**, **Endereco**, **StatusPedido**, e **ItemPedido** que vai ser criada.

Também a criação da entidade **ItemPedido** com seus relacionamentos

```

14  @Entity
15  @Data
16  @EqualsAndHashCode(onlyExplicitlyIncluded = true)
17  public class Pedido {
18
19      @Id
20      @EqualsAndHashCode.Include
21      @GeneratedValue(strategy = GenerationType.IDENTITY)
22      private Long id;
23
24      @Column(nullable = false, name = "subTotal")
25      private BigDecimal subTotal;
26
27      @Column(nullable = false)
28      private BigDecimal taxaFrete;
29
30      @Column(nullable = false)
31      private BigDecimal valorTotal;
32
33      @CreationTimestamp
34      @Column(columnDefinition = "DateTime", nullable = false)
35      private LocalDateTime dataCriacao;
36
37      @Column(columnDefinition = "DateTime")
38      private LocalDateTime dataConfirmacao;
39
40      @Column(columnDefinition = "DateTime")
41      private LocalDateTime dataCancelamento;
42
43      @Column(columnDefinition = "DateTime")
44      private LocalDateTime dataEntrega;
45
46      @ManyToOne
47      @JoinColumn(nullable = false)
48      private FormaPagamento formaPagamento;

```



```

49
50     @ManyToOne
51     @JoinColumn(nullable = false)
52     private Restaurante restaurante;
53
54     @ManyToOne
55     @JoinColumn(name = "usuario_cliente_id", nullable = false)
56     private Usuario cliente;
57
58     @Embedded
59     private Endereco enderecoEntrega;
60
61     @Column(nullable = false, columnDefinition = "varchar(10)")
62     private StatusPedido status;
63
64     @OneToMany(mappedBy = "pedido")
65     private List<ItemPedido> itensPedido = new ArrayList<>();
66 }
67

```

ItemPedido

```

ItemPedido.java x
9     @Entity
10     @Data
11     @EqualsAndHashCode(onlyExplicitlyIncluded = true)
12     public class ItemPedido {
13
14         @Id
15         @GeneratedValue(strategy = GenerationType.IDENTITY)
16         private Long id;
17
18         @Column(nullable = false)
19         private Integer quantidade;
20
21         @Column(nullable = false)
22         private BigDecimal precoUnitario;
23
24         @Column(nullable = false)
25         private BigDecimal precoTotal;
26
27         private String observacao;
28
29         @JoinColumn(nullable = false)
30         @ManyToOne
31         private Produto produto;
32
33         @JoinColumn(nullable = false)
34         @ManyToOne
35         private Pedido pedido;
36
37     }

```

Podemos gerar scripts DDL através do Schema-Generation

```
#Os 2 comandos abaixo deverão ser executados apenas 1 vez p/ criação DDL, depois comentados.

#Apenas para criar o script sql a partir das entidades mapeadas
#spring.jpa.properties.javaax.persistence.schema-generation.scripts.action=update

#Definindo o caminho do script
#spring.jpa.properties.javaax.persistence.schema-generation.scripts.create-target=src/main/resources/schema-generation/ddl-table-pe
```

se usarmos o valor update no gerador de script, será criado um arquivo apenas tabelas adicionadas posteriormente

Dados tratados:

create table pedido

```
(
  id          bigint    not null auto_increment,
  subtotal    decimal(10,2) not null,
  taxa_frete  decimal(10,2) not null,
  valor_total decimal(10,2) not null,

  restaurante_id  bigint    not null,
  usuario_cliente_id bigint    not null,
  forma_pagamento_id bigint    not null,

  endereco_cidade_id bigint    not null,
  endereco_cep      varchar(9)  not null,
  endereco_logradouro varchar(100) not null,
  endereco_numero   varchar(20) not null,
  endereco_complemento varchar(60) null,
  endereco_bairro    varchar(60) not null,

  status      varchar(10) not null,
  data_criacao datetime    not null,
  data_confirmacao datetime null,
  data_cancelamento datetime null,
  data_entrega datetime    null,

  primary key (id),

  constraint fk_pedido_endereco_cidade foreign key (endereco_cidade_id) references
cidade (id),
  constraint fk_pedido_restaurante foreign key (restaurante_id) references
restaurante (id),
  constraint fk_pedido_usuario_cliente foreign key (usuario_cliente_id) references
usuario (id),
  constraint fk_pedido_forma_pagamento foreign key (forma_pagamento_id)
references forma_pagamento (id)
) engine = InnoDB
default charset = utf8;
```

create table item_pedido

```
(
```

```
id      bigint      not null auto_increment,
quantidade smallint  not null,
preco_unitario decimal(10, 2) not null,
preco_total  decimal(10, 2) not null,
observacao  varchar(255) null,
pedido_id   bigint   not null,
produto_id  bigint   not null,

primary key (id),
unique key uk_item_pedido_produto (pedido_id, produto_id),

constraint fk_item_pedido_pedido foreign key (pedido_id) references pedido (id),
constraint fk_item_pedido_produto foreign key (produto_id) references produto (id)
) engine = InnoDB
default charset = utf8;
```

- ★ Não podemos adicionar anotações Join em atributos mapeados com mappedBy.
É interessante instanciar coleções na hora da definição.