04

Atualizando o banco com Evoluções

Vamos agora começar a criar a parte do sistema referente a uso do cliente, como cadastro, validação, confirmação, login e um painel de usuário, começando por um esboço do modelo de usuário.

```
package models;
import javax.persistence.*;
import com.avaje.ebean.Model;
@Entity
public class Usuario extends Model {
    @Id @GeneratedValue
    private Long id;
    private String nome;
}
```

Mas e como isso se reflete no banco? Bom, a gente já viu rapidamente que o Play! utiliza um sistema de evoluções para criar nossas tabelas automagicamente! Atualizando a página vemos esse sistema sendo executado, sugerindo mudanças no banco para criar a tabela de usuários. Mas repare que uma das linhas indica um drop na tabela de **Produtos**, e não queremos que isso aconteça. Então para tudo, vamos pausar o nosso cadastro de usuário e aprender como lidar com as Evolutions!

Em desenvolvimento, o *Play!* gera por padrão o arquivo `conf/evolutions/default/1.sql`, que atualiza os comandos *SQL* `create table` da seção UPs toda vez que reconhece uma mudança nos modelos. Porém existe um sistema pronto de reversão, a seção DOWNs. O padrão do *Play!* é que ele rode os DOWNs toda vez que alguma alteração tenha sido feita nos UPs, fazendo com que percamos os dados que já estiverem cadastrados.

Mas tem um jeito de contornar isso! No próprio arquivo `1.sql` as primeiras duas linhas são comentários que indicam para apagá-los se quisermos desabilitar a geração automática de SQL. Fazendo isso, temos que começar a criar o código **SQL** manualmente. Assim, as mudanças ficam separadas e reversíveis. Vamos criar uma evolução então. Remova os dois primeiros comentários, o código de criação e o código de drop da tabela de usuário do arquivo `1.sql`, resultando no conteúdo a seguir.

```
# --- !Ups
create table produto (
  id                            bigint auto_increment not null,
  titulo                        varchar(255),
  codigo                        varchar(255),
  tipo                          varchar(255),
  descricao                     varchar(255),
  preco                         double,
  constraint pk_produto primary key (id)
);
# --- !Downs
drop table if exists produto;
```

Crie então o arquivo `2.sql` com o seguinte conteúdo.

```
# --- !Ups
create table usuario (
  id    bigint auto_increment not null,
  nome  varchar(255),
  constraint pk_usuario primary key(id)
);
# --- !Downs
drop table usuario;
```

Acesse qualquer página do servidor e repare que agora a sugestão de evolução reflete somente a criação da tabela de usuários. Vamos adicionar outros campos no modelo de usuário e criar outra evolução então!

```
package models;
import javax.persistence.*;
import com.avaje.ebean.Model;
@Entity
public class Usuario extends Model {
    @Id @GeneratedValue
    private Long id;
    private String nome;
    private String email;
    private String senha;
}
```

Crie também o arquivo `conf/evolutions/default/3.sql` com o conteúdo a seguir para criar as novas colunas na tabela.

```
# --- !Ups
alter table usuario add column email varchar(255);
alter table usuario add column senha varchar(255);
# --- !Downs
alter table usuario drop column email;
alter table usuario drop column senha;
```

Temos uma grande vantagem agora de não perdemos mais nosso trabalho ao alterar os modelos, mas temos também uma desvantagem de precisarmos criar o código SQL para gerar as mudanças do banco.