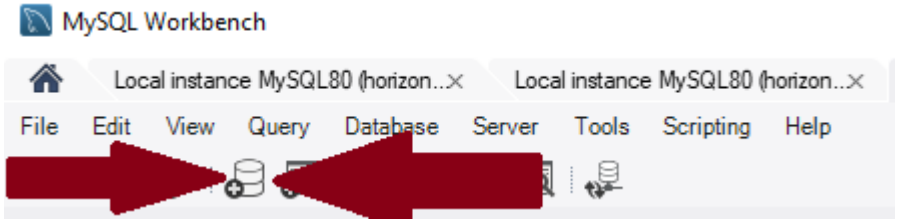
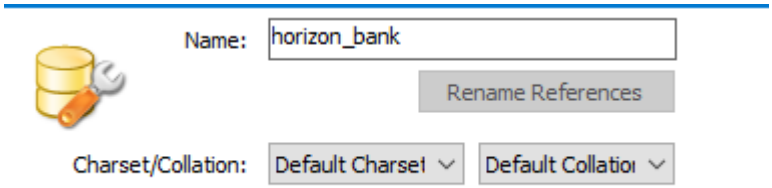
Documentação Horizon Bank

1 – Abra seu MySQL e clique nessa opção



2 – Digite esse nome e aplique



3 – Altere o username e password para os configurados por você

```yaml
spring:
  main:
    allow-circular-references: true
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/horizon_bank?useTimezone=true&serverTimezone=UTC
    username: root
    password: 1234
  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL8Dialect
    show-sql: false
```

Pessoa Service

1 – Método para criação de pessoa com validação de telefone, cpf e exception personalizada

```
1 usage
public Pessoa createPerson(PessoaDTO pessoaDTO){
    verifyTelefone(pessoaDTO.getTelefone());
    if(!verifyCpf(pessoaDTO.getCpf())){
        Pessoa pessoa = new Pessoa();
        pessoa.setNome(pessoaDTO.getNome());
        pessoa.setCpf(pessoaDTO.getCpf());
        pessoa.setTelefone(pessoaDTO.getTelefone());
        return repository.save(pessoa);
    }
    throw new CpfAlreadyExistsException();
}
```

2 – Método para verificar se telefone já foi cadastrado

```
public void verifyTelefone(String telefone){
    if(repository.findByTelefone(telefone).isPresent()){
        throw new PhoneAlreadyRegisteredException();
    }
}
```

3 – Método para verificar buscar Pessoa por CPF

```
public Pessoa findByCpf(String cpf){
    return repository.findByCpf(cpf).orElseThrow(PersonNotFoundException::new);
}
```

4 – Método para verificar se CPF já foi cadastrado

```
public Boolean verifyCpf(String cpf){
    return repository.findByCpf(cpf).isPresent();
}
```

Conta Service

1 - Método para criação de conta com validação de telefone, cpf, pessoa e exception personalizada

```
public Conta createConta(ContaDTO contaDTO){
    if(pessoaService.verifyCpf(contaDTO.getCpf())) {
            Pessoa pessoa = pessoaService.findByCpf(contaDTO.getCpf());
            if(!verifyConta(pessoa, contaDTO)){
            Conta conta = new Conta();
            conta.setPessoa(pessoa);
            conta.setSaldo(0.0);
            conta.setTipoConta(typeConta(contaDTO.getTipoConta()));
            conta.setNumero(generatingAccountNumber());
            conta.setDigito(1);
            return repository.save(conta);
        }
    }
    throw new PersonNotFoundException();
}
```

2 – Método para depositar saldo, com validação de saldo

```
public Conta depositSaldo(ContaDTO contaDTO){
    verifySaldo(contaDTO.getSaldo());
    Conta conta = findContaByNumero(contaDTO.getNumero());
    conta.setSaldo(conta.getSaldo() + contaDTO.getSaldo());
    return repository.save(conta);
}
```

3 – Método para sacar saldo, com validação de saldo

```
public Conta withdrawSaldo(ContaDTO contaDTO){
    verifySaldo(contaDTO.getSaldo());
    Conta conta = findContaByNumero(contaDTO.getNumero());
    if(conta.getSaldo() < contaDTO.getSaldo()){
        throw new InsufficientFundsException();
    }
    conta.setSaldo(conta.getSaldo() - contaDTO.getSaldo());
    return repository.save(conta);
}
```

4 – Método para consultar saldo da conta

```
public String consultSaldo(ContaDTO contaDTO){
    Conta conta = findContaByNumero(contaDTO.getNumero());
    return "Saldo: R$" + conta.getSaldo();
}
```

5 – Método de verificação de saldo negativo

```
public void verifySaldo(Double saldo){
    if(saldo < 0){
        throw new InsufficientFundsException();
    }
}
```

6 – Método para buscar conta pelo numero da conta e com excpetion personalizada caso não encontre

```
public Conta findContaByNumero(Long id){
    if(repository.findByNumero(id).isPresent()){
        return repository.findByNumero(id).get();
    }
    throw new AccountNotFoundException();
}
```

7 – Método para verificar quantas contas e tipos de contas a pessoa possui

```
public boolean verifyConta(Pessoa pessoa, ContaDTO contaDTO){
    List<Optional<Conta>> list = repository.findByPessoa(pessoa);
    int size = list.size();
    if(size == 0){
        return false;
    }
    size--;
    for(int x = size; x >= 0; x--){
        if(list.get(x).get().getTipoConta().getSigla().equals(contaDTO.getTipoConta())){
            throw new AccountTypeAlreadyExistsException();

        }
    }
    return false;
}
```

8 – Método para definir tipo de conta

```
public Tipo typeConta(String tipoConta){
    if(tipoConta.equals("C")) {
        return Tipo.CORRENTE;
    }else if(tipoConta.equals("P")){
        return Tipo.POUPANCA;
    }
    throw new TypeNotFoundException();
}
```

9 – Método para gerar número da conta

```
public Long generatingAccountNumber(){
    Double randomNumber = Math.random();
    String stringNumber = String.valueOf(randomNumber);
    int size = stringNumber.length();
    int reference = stringNumber.indexOf(".");
    String numberAccount = stringNumber.substring(reference + 1, size);
    return Long.valueOf(numberAccount);
}
```

Transferencia Service

1 – Método para realizar transferencia com validação de valor e conta de origem e destino

```java
public Transferencia transferencia(TransferenciaDTO transferenciaDTO){
    contaService.verifySaldo(transferenciaDTO.getValor());
    Conta contaOrigem = contaService.findContaByNumero(transferenciaDTO.getNumeroOrigem());
    Conta contaDestino = contaService.findContaByNumero(transferenciaDTO.getNumeroDestino());

    if(contaOrigem.getSaldo() < transferenciaDTO.getValor()){
        throw new InsufficientFundsException();
    }
    consultTransferencia(contaOrigem, contaDestino, transferenciaDTO.getValor());
    return organizingTransferencia(contaOrigem, contaDestino, transferenciaDTO.getValor());
}
```

2 – Método para organizar e salvar a transferencia no banco de dados

```java
public Transferencia organizingTransferencia(Conta contaOrigem, Conta contaDestino, Double valor){
    Transferencia transferencia = new Transferencia();
    transferencia.setData(dataFormat());
    transferencia.setValor(valor);
    transferencia.setContaOrigem(contaOrigem);
    transferencia.setContaDestino(contaDestino);
    return repository.save(transferencia);
}
```

3 – Método para pegar data da transferencia e formatar para melhor visualização

```java
public String dataFormat(){
    SimpleDateFormat formatter = new SimpleDateFormat( pattern: "dd/MM/yyyy");
    Date now = new Date();
    return formatter.format(now);
}
```

4 – Método para descontar saldo da conta origem e adicionar saldo na conta destino

```java
public void consultTransferencia(Conta contaOrigem, Conta contaDestino, Double valor){
    contaOrigem.setSaldo(contaOrigem.getSaldo() - valor);
    contaDestino.setSaldo(contaDestino.getSaldo() + valor);
    contaService.save(contaDestino);
    contaService.save(contaOrigem);
}
```

Pessoa Controller

1 – Método Post para criação de pessoa

```
no usages
@PostMapping("/create")
public Pessoa createPerson(@RequestBody PessoaDTO pessoaDTO) { return service.createPerson(pessoaDTO); }
```

Conta Controller

1 – Método Post para criação de conta

```
no usages
@PostMapping("/create")
public Conta createConta(@RequestBody ContaDTO contaDTO) { return service.createConta(contaDTO); }
```

2 – Método Get para consultar saldo

```
no usages
@GetMapping("/balance")
public String consultSaldo(@RequestBody ContaDTO contaDTO) { return service.consultSaldo(contaDTO); }
```

3 – Método Patch para depositar saldo na conta

```
no usages
@PatchMapping("/deposit")
public Conta depositSaldo(@RequestBody ContaDTO contaDTO) { return service.depositSaldo(contaDTO); }
```

4 – Método Patch para sacar saldo da conta

```
no usages
@PatchMapping("/withdraw")
public Conta withdrawSaldo(@RequestBody ContaDTO contaDTO) { return service.withdrawSaldo(contaDTO); }
```

Transferencia Controller

1 – Método Post para realizar transferencia

```
no usages
@PostMapping("/perform")
public Transferencia createTransferencia(@RequestBody TransferenciaDTO transferenciaDTO){
    return service.transferencia(transferenciaDTO);
}
```