Se você já trabalhou com algum banco de dados relacional deve ter reparado que em nenhum momento chamamos métodos como `commit` ou `rollback` para consolidar a transação ou abortá-la. Por mais que isso possa lhe causar certo espanto, o IndexedDB trabalha um pouquinho diferente.

Transações do IndexedDB são auto commited

É por meio de uma transação que temos acesso a uma store e dela podemos realizar operações como a inclusão de um objeto. Quando essa operação é realizada com sucesso, ou seja, quando o evento `onsuccess` é chamado a transação é fechada, ou seja, as transações do IndexedDB são *auto commited*. É por isso que cada método do nosso `NegociacaoDao` solicita uma transação toda vez que é chamado.

Podemos cancelar uma transação através do método `abort`

Ótimo, já sabemos quando uma transação é efetivada e que este é um processo automático, no entanto nem sempre queremos efetivá-la, ou seja, queremos abortá-la. Fazendo uma alusão aos bancos de dados relacionais, queremos ser capazes de realizar um `rollback`.

Para cancelarmos (rollback) uma transação podemos chamar o método `abort`:

```
ConnectionFactory.
    .getConnection()
    .then(connection => {

            let transaction = connection.transac

            let store = transaction.objectStore(

            let negociacao = new Negociacao(new

            let request = store.add(negociacao);

            // #### VAI CANCELAR A TRANSAÇÃO. O
            transaction.abort();

            request.onsuccess = e => {

                console.log('Negociação incluida
            };

            request.onerror = e => {

                console.log('Não foi possível in
            };


    })
```

Ao executar o código a seguinte mensagem de erro será exibida no console:

```
DOMException: The transaction was aborted, so th
Não foi possível incluir a negociação
```

Trate o cancelamento de uma transação no evento `onabort` de transaction

Contudo, podemos tratar os erros de uma transação abortada no evento `onabort` da transação, ao invés de lidarmos com ele em `onerror`.

```
ConnectionFactory.
    .getConnection()
    .then(connection => {

            let transaction = connection.transac

            let store = transaction.objectStore(

            let negociacao = new Negociacao(new

            let request = store.add(negociacao);

            // #### VAI CANCELAR A TRANSAÇÃO. O

            transaction.abort();
            transaction.onabort = e => {
                console.log(e);
                console.log('Transação abortada'
            };

            request.onsuccess = e => {

                console.log('Negociação incluida
            };

            request.onerror = e => {

                console.log('Não foi possível in
            };


    })
```

Apesar do que aprendemos aqui não ser útil dentro do cenário da aplicação Aluraframe, informações extras como essa são sempre bem-vindas!