Create Linked Document

Scenario

In the realm of duelists, when a `Duelist` is created, a button shall appear, allowing the creator to forge a new `Deck`. Upon invoking this button, a new `Deck` document shall be created, with the `Duelist`'s name pre-filled. However, the server must validate that each `Duelist` can only have one `Deck`.

Client Script

File Path:

```
[your_app]/[your_app]/doctype/duelist/duelist.js
```

Code:

```
frappe.ui.form.on('Duelist', {
    refresh: function(frm) {
        // Check if the Duelist has materialized in the database
    // frm.doc.__islocal == true when the document is new (not in the db yet, still in creation)
        if (!frm.doc.__islocal) { // Make sure the button does not appear when the Duelist has not yet been creat
            // If true, conjure the "Build a New Deck" button
            frm.add_custom_button(__('Build a New Deck'), function() {
                // Logic to create a new Deck
                frappe.new_doc('Deck', {
                    duelist: frm.doc.name //use the PK to bind/link
                });
            });
        }
    }
});
```

Explanation:

- `frm.add_custom_button`: Adds a button to the `Duelist` form.
- `frappe.new_doc`: Creates a new `Deck` document with the `Duelist`'s name pre-filled.

Server Script

This script shall reside within the `Deck` DocType, ensuring that each `Duelist` can only have one `Deck`.

File Path:

```
[your_app]/[your_app]/doctype/deck/deck.py
```

Code:

```
import frappe
from frappe.model.document import Document
from frappe import _ # The _ function is used for localization
# In JavaScript, we use __ instead to localize

class Deck(Document):
 def validate(self):
  self.validate_one_deck_per_duelist()

 def validate_one_deck_per_duelist(self):
  if frappe.db.exists('Deck', {'duelist': self.duelist, 'name': ['!=', self.name]}):
   frappe.throw(_("The duelist {0} already possesses a deck!").format(frappe.bold(self.duelist)))
```

Explanation:

- `frappe.db.exists`: Checks if there is already a `Deck` linked to the `Duelist`.
- `frappe.throw`: Prevents the creation of the new `Deck` if the condition is not met.

▼ Detailed explanation on the validate functions

The Spell Components:

1. `existing_deck = frappe.db.exists('Deck', {'duelist': self.duelist, 'name': ['!=', self.name]})`
   - `frappe.db.exists`: A potent function that checks if a document of a particular DocType exists based on certain conditions.
   - `'Deck'`: The DocType we are querying. We seek to find if a `Deck` already exists for our duelist.
   - `{'duelist': self.duelist, 'name': ['!=', self.name]}`: The conditions for our search. We are looking for a `Deck` where:
     - `'duelist': self.duelist`: The `duelist` field matches the duelist of our current document (`self.duelist`).
     - `'name': ['!=', self.name]}`: The `name` of the deck is not equal (`!=`) to the name of our current document (`self.name`). This ensures that we are not counting the current document in our check.
2. `if existing_deck:`
   - This condition checks whether the `existing_deck` variable holds a truthy value (i.e., a deck was found that matches our conditions). If it is true, the logic within the if block will be executed.
3. `frappe.throw(_("The duelist {0} already possesses a deck!").format(frappe.bold(self.duelist)))`
   - `frappe.throw`: A function that halts the current operation and displays a message to the user. When this function is invoked, the document will not be saved, ensuring our validation logic is enforced.
   - `_("The duelist {0} already possesses a deck!")`: The message that will be displayed. The `{0}` is a placeholder where a variable will be inserted using the `format` method.
   - `.format(frappe.bold(self.duelist))`: The `format` method replaces `{0}` in our message with the name of the duelist, which is made bold using `frappe.bold`.

The `validate` method is indeed invoked before a document is inserted into the database, **but it is also invoked before updates** to existing documents are committed. Thus, the `validate` method serves to guard against forbidden alterations in both new and existing documents.

An Example from the Shadow Realm:

Imagine a duelist, Yugi, who already possesses a deck named "Dark Magician's Arsenal" in the database. One day, Yugi decides to update some details of this deck. When he attempts to save the changes, the `validate` method is invoked.

- Without `'name': ['!=', self.name]`:
  - The logic searches for any deck bound to Yugi and finds "Dark Magician's Arsenal" (the very deck Yugi is trying to update). Believing this to be a separate, conflicting deck, the logic prevents the update and casts a message of despair upon Yugi.
- With `'name': ['!=', self.name]`:
  - The logic searches for any deck bound to Yugi but ignores "Dark Magician's Arsenal" during its search. Finding no conflicting decks, the logic permits the update, and Yugi's alterations are saved to the shadowy depths of the database.

The `before_insert` method is a different kind of enchantment in the Frappe framework. It is invoked only before a new document is inserted into the database, not when it is updated.

Example:

Let's say Yugi already possesses a deck named "Dark Magician's Arsenal". If he tries to create a new deck, the `before_insert` method will be invoked and can prevent the creation if a deck for Yugi already exists. However, if Yugi tries to update an existing deck, the `before_insert` method will not be invoked, and thus, the validation logic inside it will not be executed.