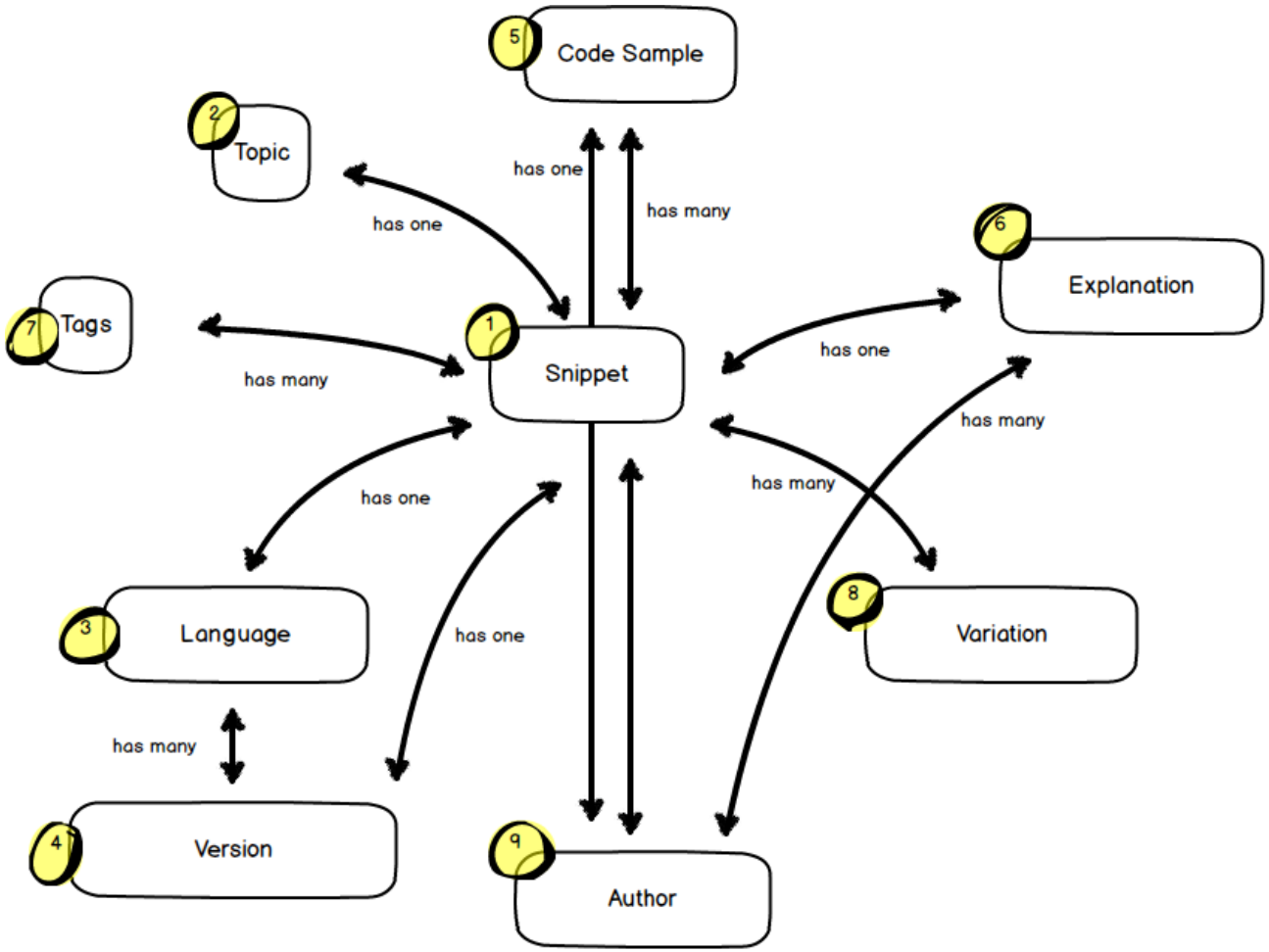



1. A snippet is a small template for code that performs a specific task in a specific language and language version.

2. Each snippet belongs to a specific topic. Examples may include: Promises, jQuery AJAX request, SQL Select statement, etc.

3. Each snippet is written for a specific programming language

4. Each snippet is written for a specific version of said programming language

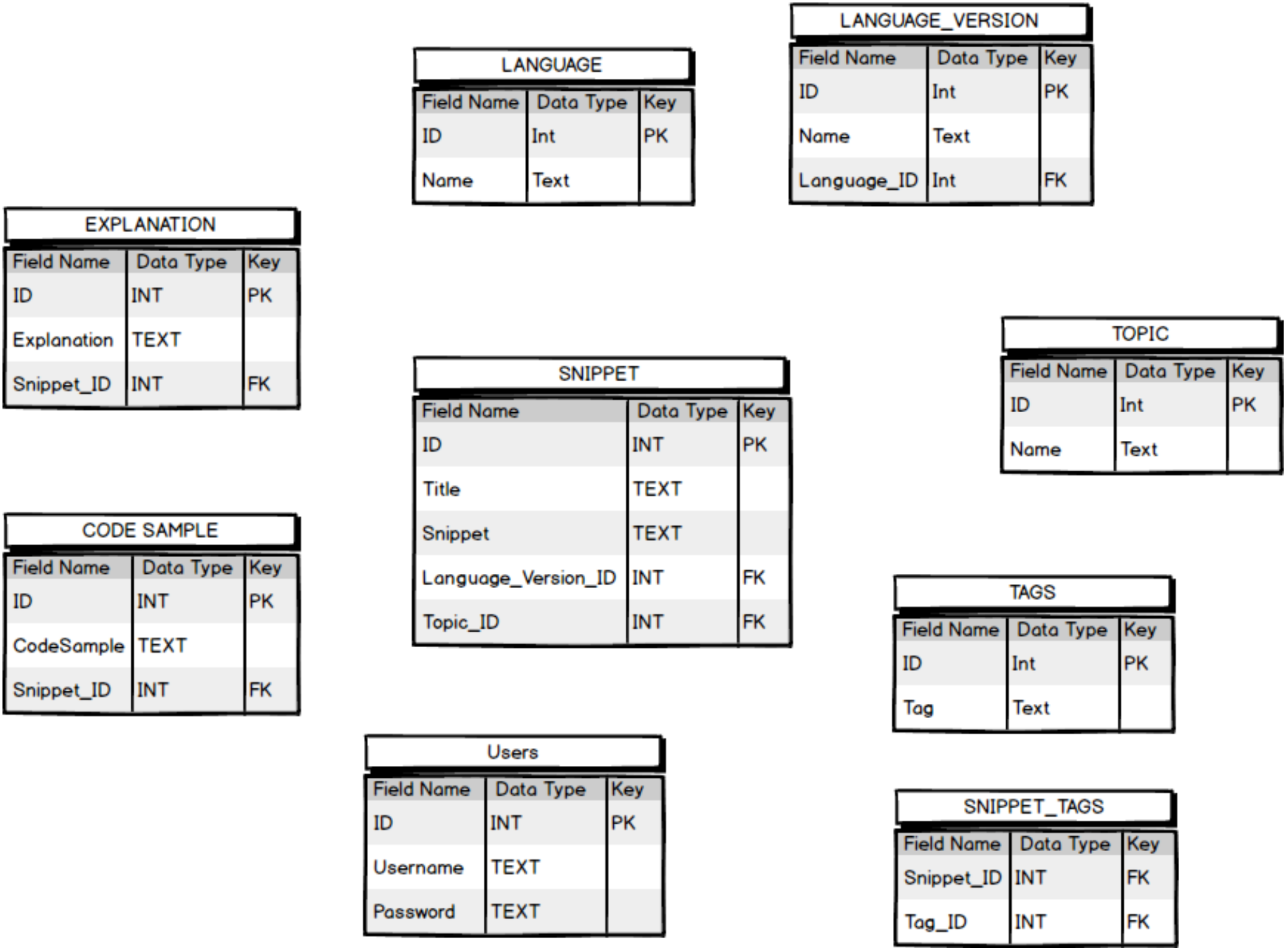
5. Snippets may be accompanied by fully written, working code samples that demonstrate the topic. A snippet may have more than one related code sample.

6. Snippets may be accompanied by an explanation of the topic. There is only one explanation allowed for each snippet.

7. Snippet authors and consumers can tag the snippet with additional keywords to enhance search

8. Sometimes there is more than one way to perform a task. In this case, a snippet can have more than one variation. All related snippets would be stored in the Snippet table, but the relationship would be tracked in another table.

9. A snippet has one author. A code sample has one author. An explanation can have many authors (like a wiki).


LANGUAGE
Field Name
Data Type
Key
ID
Int
PK
Name
Text
LANGUAGE_VERSION
Field Name
Data Type
Key
ID
Int
PK
Name
Text
Language_ID
Int
FK
EXPLANATION
Field Name
Data Type
Key
ID
INT
PK
Explanation
TEXT
Snippet_ID
INT
FK
SNIPPET
Field Name
Data Type
Key
ID
INT
PK
Title
TEXT
Snippet
TEXT
Language_Version_ID
INT
FK
Topic_ID
INT
FK
TOPIC
Field Name
Data Type
Key
ID
Int
PK
Name
Text
CODE SAMPLE
Field Name
Data Type
Key
ID
INT
PK
CodeSample
TEXT
Snippet_ID
INT
FK
TAGS
Field Name
Data Type
Key
ID
Int
PK
Tag
Text
Users
Field Name
Data Type
Key
ID
INT
PK
Username
TEXT
Password
TEXT
SNIPPET_TAGS
Field Name
Data Type
Key
Snippet_ID
INT
FK
Tag_ID
INT
FK

# Creating a Promise in JavaScript (ES6)

## Snippet

```
//Declaration

let myFirstPromise = new Promise((resolve, reject) => {
});

//Use:

myFirstPromise.then((successMessage) => {
});
```

## Examples

One | Two | Three | Four

```
let myFirstPromise = new Promise((resolve, reject) => {
  var x = 5;
  var y = 10;
  resolve(x * y);
});
myFirstPromise.then((success) => {
  console.log(success); // console logs 50
});
```

Rate this Code Sample

## Explanation

A promise can be:

fulfilled - The action relating to the promise succeeded
rejected - The action relating to the promise failed
pending - Hasn't fulfilled or rejected yet
settled - Has fulfilled or rejected

The promise constructor takes one argument, a callback with two parameters, resolve and reject. Do something within the callback, perhaps async, then call resolve if everything worked, otherwise call reject.

Like throw in plain old JavaScript, it's customary, but not required, to reject with an Error object. The benefit of Error objects is they capture a stack trace, making debugging tools more helpful.

## Tags

JavaScript Promises ES6 Asynchronous Programming