Développement d'une application web de création de quiz

Version 0.1

Benoît Léo Maillard

14 January 2015

Table des matières

CHAPITRE 1

Pourquoi utiliser Python plutôt que Javascript ?

Les sites web d'aujourd'hui utilisent de plus en plus le javascript afin de rendre la navigation plus confortable pour le visiteur. Les fonctionnalités offertes par ce language permettent de concevoir une grande variété d'applications utilisables directement dans le navigateur, qui s'éxécutent côté client et qui sont donc par conséquent très accessibles au grand public. Le Javascript est donc un langage quasiment incontournable pour toute personne qui s'intéresse au développement web.

Malgré cela, ce langage n'est pas forcément facile à aborder pour un programmeur habituée à coder en Java, en Python ou dans un autre langage de programmation moderne, car son fonctionnement diffère dans un certain nombre d'aspects fondamentaux. Par exemple, le Javascript est un langage orienté objet à prototype, c'est à dire que les objets utilisés en Javascript sont des copies d'objets prototype, et non des instances de classes comme en Python et Java. On peut également mentionner d'autres différences importantes, comme la portée des variables par défaut. Afin de palier aux difficultés que peut rencontrer un développeur qui débute dans ce langage, divers outils sont apparus pour tenter de remplacer Python par Javascript, avec pour objectif de combiner les possibilités offertes par le Javascript avec la simplicité et la clarté de Python. Ce travail a pour but de présenter et expliquer les fonctionnalités de RapydScript, un outil permettant d'écrire du code très semblable à du code Python (dans sa syntaxe et sa philosophie) et de générer du Javascript à partir de ce code.

CHAPITRE 2

Différentes manières d'aborder le problème : CoffeeScript, Brython et RapydScript

Un certain nombre d'outils open-source tentent de faciliter l'écriture de code Javascript et différentes approches du problème sont possibles. CoffeeScript (http://coffeescript.org/) est décrit par ses auteurs comme un langage à part entière, avec sa propre syntaxe (qui ressemble un peu à celle de python, mais qui s'inspire surtout de ruby), qui peut être compilé en Javascript. CoffeScript permet l'écriture d'un code source plus clair et concis, le code généré est éxecuté avec les mêmes performances que du code javascript natif (puisqu'il s'agit simplement de javascript), mais son utilisation nécessite l'apprentissage d'un nouveau langage, ce qui peut être un problème pour un débutant.

À l'opposé, Brython (http://www.brython.info/) propose une toute autre approche : le développeur peut écrire du code en Python qui sera éxecuté par un interpréteur python entièrement intégré dans la page web codé en javascript. C'est certainement la solution la plus fidèle à Python, puisque la syntaxe de Brython est exactement identique à celle de Python. L'accès au DOM et l'utilisation d'Ajax se fait via l'import de modules externes. Brython est donc idéal pour quelqu'un qui ne connaît pas Javascript mais possède de bonnes bases en Python. Cependant, comme la traduction en Javascript se fait "en live", l'impact sur les performances se fait ressentir (http://pyppet.blogspot.ch/2013/11/brython-vs-pythonjs.html) et peut poser problèmes pour des scripts complexes. Le script de l'interpréteur (environ 10'000 lignes) doit également être inclus dans chaque page html qui comporte du code Brython. De plus, il n'est pas possible d'utiliser des libraires Javascript externes, telles que jQuery ou AngularJS.

RapydScript a l'avantage de combiner les qualités des deux outils cités plus haut, en permettant d'écrire du code très similaire à du code Python standard et en le compilant en Javascript. Rapydscript est donc facile à prendre en main pour n'importe quelle personne sachant coder en Python et ne limite pas la rapidité d'éxecution puisque le code qui sera executé est tout simplement du Javascript. Il est aussi possible d'appeler n'importe quelle fonction Javascript standard et par extension d'utiliser n'importe quelle librairie externe. Ce dernier point n'est pas négligeable puisque j'ai opté pour jQuery pour développer mon application de création de quiz en ligne. Le choix de RapydScript pour la réalisation de ce travail est apparu comme évident après avoir considéré les qualités et défauts de ces différents outils. La prise en main ainsi que la compréhension de son fonctionnement a pu se faire aisément, d'autant plus que l'absence de documentation ou de tutoriel complet sur cette technologie en français constitue une motivation supplémentaire et rend ce travail inédit.

CHAPITRE 3

Introduction à RapydScript

3.1 Installation

3.2 Compilation

3.3 Programmer avec RapydScript

3.3.1 Notions de base

Pour un habitué de Python, la prise en main de RapydScript peut se faire très rapidement : il suffit d'écrire son programme comme si on écrivait un programme Python, même si dans certains cas particuliers il n'est pas possible de faire exactement la même chose avec RapydScript. Ces cas particuliers seront abordés plus loin.

Pour commencer avec un exemple simple, voici en python une fonction qui prend nombres en argument et retourne le plus grand. La fonction est ensuite appelée. Ce code est parfaitement valide en Python.

```
def maximum(n1, n2):
    if n1 >= n2:
        return n1
    elif n2 > n1:
        return n2

maximum(n1, n2) #Appel de la fonction
```

Une fois la compilation effectuée avec RapydScript, on obtient ce résultat :

```
function maximum(n1, n2) {
    if (n1 >= n2) {
        return n1;
    } else if (n2 > n1) {
        return n2;
    }
}
maximum(5, 18);
```

On peut voir les opérations qu'a fait RapydScript "traduire" le code source en Javascript : Remplacer le mot clé def par function, ajouter les accolades qui englobent la fonction et les if, ajouter des parenthèses autour des conditions, et ajouter un ; à la fin de chaque instruction. On remarque aussi que les commentaires ont été supprimés, puisqu'ils sont seulement utiles dans le code source. Le code ainsi produit peut maintenat être éxecuté dans n'importe quel navigateur.

Cet exemple montre cependant un cas assez général puisque le code Javascript correspondant au code source est quasiment identique. Mais RapydScript permet aussi de compiler du code typiqement Python, comme par exemples des boucles for, qui n'ont pas d'équivalent en javascript.

```
names_list = ["Paul", "Marie", "Pierre", "Lucie"]

for name in names_list:
    print(name)
```

RapydScript produit un code équivalent en Javascript qui s'éxecutera comme en Python. Le code généré est cette fois plus complexe et des connaissances en javascript sont nécessaires pour le comprendre. Ce type de manipulations sera étudié dans un prochain chapitre. On peut par exemple aussi implémenter une fonction avec des arguments qui prennent des valeurs par défaut, ce qui n'est pas possible en javascript.

3.3.2 Programmation orientée objet (POO)

Ce qui fait de RapydScript un outil si puissant est principalement les possibilités qu'il offre pour faire de la POO. En Javascript, la POO est basée sur le prototypage, et il est beaucoup plus complexe de créer ses propres objets, avec de l'héritage, etc. En python, cela est beaucoup plus simple, il est donc particulièrement intéressant de pouvoir utiliser la POO Python pour faire de la programmation web frond-end.

Encore une fois, il suffit d'écrire une classe comme on le ferait en Python :

```
class MyObject:
    def __init__(self, name):
        self.name = name

    def get_name(self):
        return self.name

object = MyObject("Object 1") #Instanciation avec un paramètre
object.get_name() #Retourne "Objet 1"
```

Il est également possible de faire de l'héritage :

```
class MyObjectPlus(MyObject): #Classe qui hérite de la classe créée précédemment
    def __init__(self, name, number):
        MyObject.__init__(self, name)
        self.number = number

    def get_number(self):
        return self.number

    def informations(self):
        return "Nom : " + self.name + " /// Nombre : " + str(self.number)

objet = MyObjectPlus("Objet 2", 5)
objet.get_name() #Retourne "Objet 2"
objet.get_number() #Retourne 5
objet.informations() #Retourne "Nom : Objet 2 /// Nombre : 5"
```

Il n'est par contre pas possible de définir des variables de classes avec RapydScript (si on définit une variable de classe, RapydScript l'ignore simplement). Cela est dû au fait qu'en Javascript, un objet n'est pas une instance d'une classe comme en Python, mais une copie d'un objet prototype. En Python, une variable de classe est en fait un attribut de l'objet Class. Il n'y a donc qu'une seul espace en mémoire pour cette variable. En Javascript, tous les attributs du prototype sont copiées à chaque fois qu'un objet est créé et il n'y a aucun équivalent aux variables de classe.

3.3.3 Utilisation de la bibliothèque standard Python

Avec RapydScript, il est possible d'utiliser dans le code source des fonctions provenant de différentes sources. La première est la bibilothèque standard de Python, c'est à dire les fonctions natives Python, telles que print(), len() ou range().

Pour utiliser les fonctions de la bibliothèque standard Python, il faut placer l'instruction suivante au début du fichier Python :

```
import stdlib
```

RapydScript définira ainsi ces fonctions dans le fichier généré et leur comportement sera en quelque sorte simulé en Javascript. Ces fonctions peuvent donc être utilisés comme on le ferait en Python, dans le code source.

3.3.4 Séparer son code en plusieurs fichiers

Il est déconseillé d'écrire tout son code dans un seul fichier lorsqu'on travaille sur un gros projet. Pour séparer son code en plusieurs fichiers, RapydScript prévoit un système d'imports qui ressemble à celui de Python. Voici comment procéder pour écrire son code dans plusieurs fichiers.

Premièrement, chaque fichier du code source doit utiliser l'extension .pyj, qui est l'extension des fichiers RapydScript. Ensuite, ces fichiers peuvent être utilisés comme des modules et être importés depuis un autre fichier. Par exemple, si on a placé une partie de notre code dans un fichier "moduletest.pyj", on ajoutera l'instruction suivante en début de fichier :

```
import moduletest
```

Lors de la compilation, RapydScript va rassembler tous les fichiers du code source dans un même fichier Javascript, ce qui facilite aussi l'insertion du script dans un fichier HTML. Il est important de noter, cependant, que l'import d'un module ne rend pas ses fonctions disponibles dans un namespace distinct (ce qui est le cas en Python par contre). Par exemple, pour appeler la fonction test du module de l'exemple précédent, voici comment procéder :

```
import moduletest

test() #Correct
moduletest.test() #Ne fonctionne pas
```

3.3.5 Utilisation de fonctions Javascript natives ou provenant de librairies exernes

Il est également possible d'utiliser des fonctions Javascript natives, par exemple :

```
#Ces deux expressions sont équivalentes :
console.log("Bonjour") #Fonction javascript
print("Bonjour") #Fonction python
```

L'exemple parle de lui-même et ne nécessite pas d'explications supplémentaires. Il peut parfois être pratique d'utiliser des fonctions qui n'ont pas d'équivalent en Python.

Mais une autre grande force de RapydScript est la possibilité d'utiliser des librairies Javascript externes, telles que jQuery ou AngularJS. Pour cela, rien de plus simple, il suffit d'insérer le script (par exemple jQuery) que l'on veut utiliser dans le code HTML, comme ceci :

```
<html>
<head>
    <script src="jquery.js"></script><!-- jQuery -->
    <script src="myscript.js"></script><!-- Script créé avec RapydScript -->
</head>
<body>
    <div id="mydiv"></div>
</body>
</html>
```

On peut maintenant utiliser les fonctions jQuery dans notre code. Cette fonction sélectionne le et y insère du texte :

```
def add_text(text):
    $("#mydiv").text(text)

add_text("Hello World")
```

On peut procéder de la même manière pour n'importe quelle autre librairie externe Javascript.

3.3.6 Débugger avec RapydScript

3.3.7 Cas problématiques

CHAPITRE 4

Comparaison du code source et du code généré

CHAPITRE 5

Description des fonctionnalités de l'application

5.1 Professeurs

5.1.1 Création de quiz

Les professeurs peuvent créer un quiz en utilisant un format texte de type markdown pour créer différents types de questions et y attribuer certaines caractéristiques. Cet éditeur de quiz se présente se présente sous la forme d'une zone de texte dans laquelle le professeur peut entrer le code du quiz. Au début de chaque ligne, le prof entre un tag, chaque tag a une signification différente. Chaque tag doit être séparé de la suite de la ligne par un espace

Tag	Signification
##	Question à choix multiple avec plusieurs options cochables
**	Question à choix multiple avec une seule option cochable
^^	Liste déroulante avec une option sélectionnable
? ?	Question avec un champ de texte à remplir
*	Option invalide dans un QCM
=	Option valide dans un QCM
=	Réponse correcte dans une question à champ de texte
.	Permet de définir le nombre de points sur la question (par défaut, 1)
+	Ajout d'un commentaire d'explication qui sera affiché lors de la correction

Voici un exemple de quiz comprenant 4 questions :

Lorsque le professeur appuie sur le bouton "Aperçu", il peut voir le rendu du quiz tel que le verront les étudiants. Les éventuelles erreurs dans le code (tag qui n'existe pas, etc.) sont affichés en rouge en dessous de l'aperçu. Le quiz ne peut pas être enregistré tant qu'il y a encore des erreurs dans le code. S'il commence à créer un quiz et désire le terminer plus tard, le professeur peut enregistrer un brouillon. L'outil de création de quiz supporte l'affichage des mathématiques avec MathJax (http://www.mathjax.org/).

Le prof doit également donner un titre au quiz et peut le relier à un plusieurs chapitres. Il décide de restreindre l'accès aux membres d'un ou plusieurs groupes ou de le rendre public. Pour créer un quiz, un prof a aussi la possibilité de reprendre un quiz déjà existant et de modifier son code. S'il opte pour cette solution, les statistiques du nouveau quiz ainsi créé repartent à zéro, puisque les questions ne sont plus forcément les mêmes.

5.1.2 Consultation des statistiques avancées

Après que les élèves d'un groupe ont complété un quiz, le professeur peut afficher le résultat moyen des élèves, le % de réussite à chaque question, mais aussi les réponses soumises par chaque élève en particulier. Le prof peut ainsi se faire une idée globale et plus cibleé du niveau de compréhension de son groupe.

5.2 Étudiants

5.2.1 Trouver un quiz

Pour trouver un quiz, un étudiant a plusieurs possibilités. Le professeur peut donner l'url exacte du quiz à compléter, ce qui peut être pratique dans un email ou toute communicaton informatisée. Une fonctionnalité permet au utilisateurs de générer un code QR correspondant à un quiz, ce qui est idéal pour afficher sur un projecteur en classe ou sur une feuille imprimée. Un étudiant peut aussi accéder à un quiz en mémorisant son id et en l'entrant dans la champ prévu à cet effet sur la page "Trouver un quiz".

Dans son espace utilisateur, l'étudiant peut aussi consulter les derniers quizs créés dans son groupe et peut donc y accéder facilement.

5.2.2 Compléter un quiz et correction automatique

Une fois que l'étudiant à accédé au quiz, il peut le compléter très simplement en remplissant les champs de formulaires affichés. Lorsqu'il a fini, il peut soumettre ses réponses et est redirigé vers une page de correction. Les réponses incorrectes sont affichées en rouge avec la solution et une explication pour chaque question. Les points reçus pour chaque question sont affichés avec le total de points sur le quiz. L'étudiant peut aussi comparer son score à la moyenne des autres étudiants du groupe. Un champ de texte est disponible pour envoyer des éventuelles remarques au professeur (signaler une erreur, poser une question). La page pour compléter un quiz ainsi que celle de la correction sont optimisées pour mobile et le design responsive s'adapte parfaitement à tous types de périphériques (desktop, portable, tablette, téléphone mobile).

CHAPITRE 6

Organisation du code côté serveur

CHAPITRE 7

Organisation du code côté client

CHAPITRE 8

Modèle relationnel

8.1 Introduction

Une des premières étapes importantes dans le développement d'un site web est d'imaginer un modèle relationnel structuré permettant de stocker toutes les données générées par l'application et de les relier entre elles. Un modèle relationnel décrit les différentes tables de la base de donnée et les liens entre ces tables. Une table peut être comparée à un tableau contenant des informations. Chaque table comporte une ou plusieurs colonnes, chaque colonne stockant un type de données précisément défini (par exemple, un nombre entier ou une chaîne de caractères). Si on imagine une table contenant les données sur les utilisateurs d'un site, une colonne pourrait alors définir le pseudonyme d'un utilisateur et une autre son âge. On peut ensuite ajouter des lignes à une table, c'est à dire un ensemble de données dont chaque élément correspond à une colonne de la table. Dans l'exemple précédent, on ajouterait ainsi une ligne pour chaque utilisateur s'inscrivant sur le site. Chaque ligne est identifiée grâce à une clé primaire unique, habituellement sous la forme d'un entier, qui permet de créer des liens entre différentes lignes de différentes tables (appelées relations).

Voici, présenté sous forme simplifiée à l'aide d'un tableau, comment on pourrait stocker les données concernant des quiz ainsi que leurs créateurs :

Table contenant les utilisateurs :

Clé primaire	Prénom	Âge
1	Paul	26
2	Juliette	22
3	Marc	48

Table contenant les quiz :

Clé primaire	Titre du quiz	Nombre de questions	Auteur
1	Fonctions exponentielles	5	3
2	Logarithmes	9	3
3	Comportement à l'infini	8	2

Ici, le quiz #1 a comme titre *Fonctions exponentielles*, comporte 5 questions et a été créé par Marc (Utilisateur 3).

Une fois que le modèle relationnel a été élaboré, on peut créer une base de données sous forme informatique. Le langage SQL permet de créer les tables d'une base de données, d'y enregistrer des informations et de faire des requêtes (c'est à dire récupérer des données enregistrées). Un logiciel ou une application web peut ainsi communiquer avec une base de données et stocker les informations nécessaires de manière permanente.

Django offre la possibilité de créer une base de données et d'interagir celle-ci par le biais d'une API, on peut donc éviter l'utilisation du langage SQL et communiquer avec la base de données avec des objets et des méthodes Python.

8.2 Utilisation dans l'application de création de quiz

8.2.1 Implémentation dans Django

Comme indiqué précédemment, Django fournit une interface de haut niveau pour créer et interagir avec une base de données. On peut écrire nos tables avec une architecture orientée objet, que Django "traduira" ensuite en SQL. Ainsi, pour créer une table dans notre base de données, il suffit de définir une classe héritant de `models.Model` et d'initialiser des variables de classes pour ajouter des colonnes à la tables. Ainsi, par exemple, pour implémenter une table contenant les données générales d'un quiz, il suffit d'écrire le code suivant :

```
class Quiz(models.Model): #Infos générales sur le quiz
    title = models.CharField(max_length=100) #Colonne contenant une chaîne de caractères de longueur
    creation_date = models.DateTimeField() #Colonne contenant une date/heure
    code = models.CharField(max_length=1000) #Colonne contenant une chaîne de caractères de longueur
```

Les fonctions comme `CharField()` ou `DateTimeField()` permettent de définir des champs d'un type de données précis. Une liste complète des types de champs est disponible sur la documentation officielle de Django : https://docs.djangoproject.com/en/1.7/ref/models/fields/#field-types

8.2.2 Schéma global

L'élaboration d'un schéma relationnel n'est pas chose facile car il est nécessaire que celui ci réponde à certains critères. Il doit être relativement simple afin de garder une certaine flexibilité et d'avoir la possiblité d'être modifié plus tard, car il est souvent difficile de prédire à l'avance les difficultés liées au stockage des données qui pourraient être rencontrées au cours du développement. Il doit également être possible d'ajouter des fonctionnalités sans devoir revoir toute l'organisation du schéma ou créer un trop grand nombre de nouvelles tables. Malgré cela, le modèle doit aussi correspondre aux exigences des fonctionnalités de l'application et doit inclure toutes les relations nécessaires. Il s'agit donc d'une étape cruciale qui peut s'avérer décisive pour la suite du développement.

Voici le diagramme des tables utilisés pour stocker les données des quiz :

8.2.3 Explications des tables

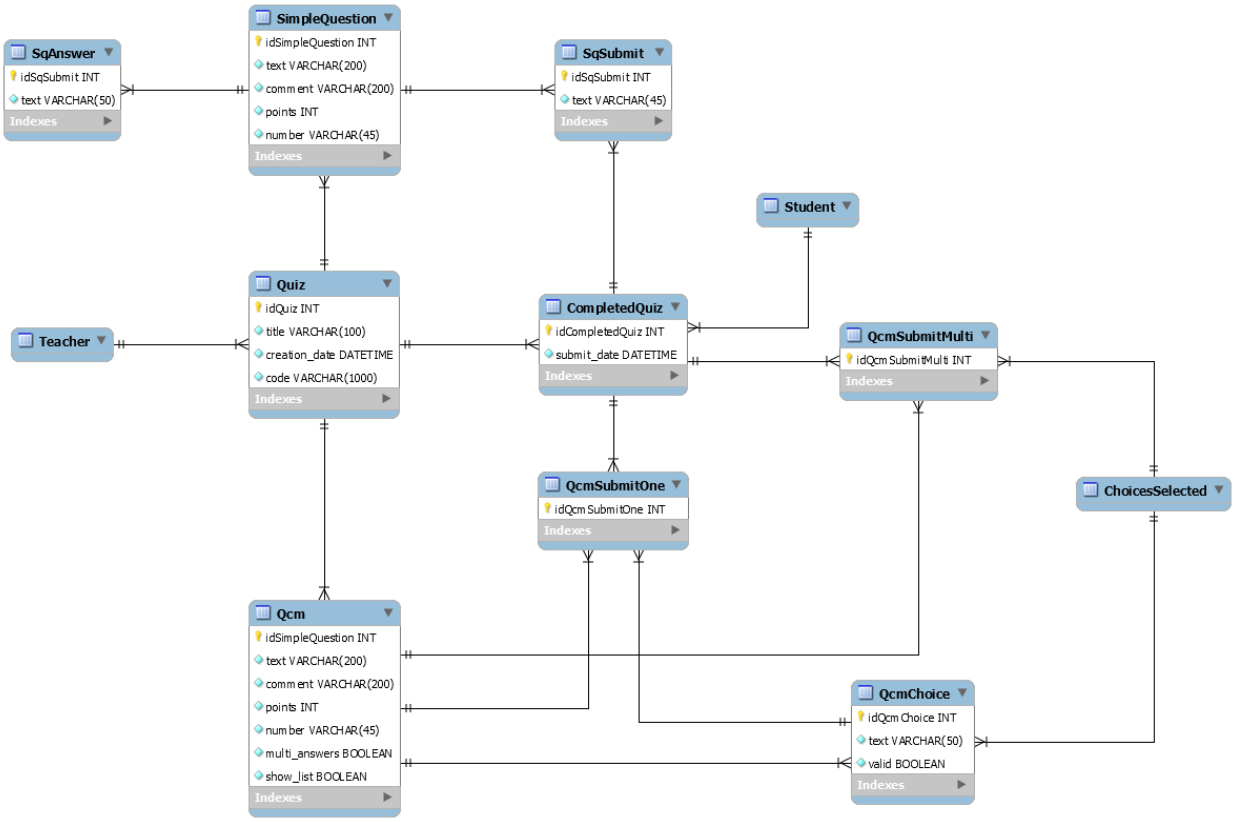
Quiz

La table *Quiz* est la table centrale de l'application et toutes les autres tables s'organisent autour d'elle. Elle comporte trois colonnes : une contenant le titre, une autre contenant la date et l'heure de création (ajouté automatiquement lorsqu'un nouveau quiz est créé) ainsi qu'une colonne dans laquelle est stockée le code du quiz. L'intérêt de stocker ce code n'est pas forcément évident. Il est stocké afin de permettre à un professeur de créer un nouveau quiz à partir d'un quiz créé précédemment par lui-même ou un autre professeur (à condition que celui-ci l'ait rendu public). Le code peut ainsi être modifé pour ajouter des questions ou apporter une quelconque modification. Cette table comporte également une relation vers une table Teacher (qui ne fait pas partie de l'application de quiz) qui permet l'intégration dans le projet de groupe.

— **Note :** la fonction de récupération de quiz existants n'est pas encore fonctionnelle.

Avec django, il est possible d'initialiser automatiquement un champ `DateTimeField` à la date/heure du moment où le modèle est instancié avec le paramètre `auto_now_add`

```
creation_date = models.DateTimeField(auto_now_add=True)
```



SimpleQuestion

Cette table contient les informations générales sur les questions simples du quiz. Ces questions sont présentées sous la forme d'un simple champ de texte lorsqu'un élève complète le quiz. Une première colonne *title* stocke l'énoncé de la question, *comment* permet d'inclure un commentaire affiché lors de la correction automatique du quiz (par exemple la démonstration d'une égalité), *points* définit le nombre de points attribués sur cette question et *number* enregistre l'ordre auquel doit apparaître la question dans le quiz. Une relation désigne le quiz qui intègre la question.

SqAnswer

Cette table contient simplement la solution de la question définie par la relation vers la table *SimpleQuestion*. Il est important de noter qu'il peut y avoir plusieurs solutions possibles pour une question et c'est la raison pour laquelle la solution n'est pas simplement stockée dans une colonne de *SimpleQuestion*.

Qcm

La table Qcm permet de stocker les informations générales à propos des questions à choix multiples. Ces questions sont affichées sous forme de boutons radio, de cases à cocher ou de liste déroulante. Cette table reprend plusieurs colonnes de la table *SimpleQuestion*. C'est pourquoi ces deux tables héritent en fait du même modèle dans django :

```
class QuizQuestion(models.Model): #Classe abstraite dont héritent tous les types de questions
    text = models.CharField(max_length=200) #Énoncé
    comment = models.CharField(max_length=200, blank=True) #Commentaire affiché lors de la correctio
    points = models.FloatField(default=1)
    number = models.IntegerField() #Ordre de la question dans le quiz
```

```
    id_quiz = models.ForeignKey(Quiz)

    class Meta:
        abstract = True

class SimpleQuestion(QuizQuestion):
    pass #Cette table reprend simplement les mêmes colonnes que le modèle abstrait

class Qcm(QuizQuestion):
    multi_answers = models.BooleanField() #True si il est possible de cocher plusieurs choix
    show_list = models.BooleanField() #True si les choix sont affichés sous forme de liste déroulante
```

En plus des colonnes héritées, *Qcm* possède deux champs de types booléens : *multi_answers*, qui définit si plusieurs options peuvent être cochées ou non, et *show_list*, qui vaut `True` si la question est affichée sous forme de liste déroulante. Si plusieurs options peuvent être cochées, la question sera dans tous les cas affichée à l'aide de cases à cocher.

QcmChoice

Cette table contient les différents choix possibles pour la question définie par la relation vers *Qcm*. Elle est formée de deux champs, le premier contenant le texte du choix et l'autre définissant par un booléen s'il est correct ou non de cocher ce choix. Une question à choix multiples doit avoir au moins deux choix possibles et au moins un choix correct. Si *multi_answers* vaut `False` dans *Qcm*, une seule option peut être correcte puisque l'étudiant n'a la possibilité de cocher qu'une seule option.

— **Note :** D'un point de vue purement relationnel, comme il est indiqué sur le diagramme, cette table possède une relation vers une table qui sert d'intermédiaire entre *QcmChoice* et *QcmSubmitMulti*. Cette table intermédiaire crée en fait une relation de type *many-to-many*. L'implémentation de ce type de relation avec Django sera abordé plus loin.

CompletedQuiz

Comme on peut le voir sur le diagramme, à l'instar de *Quiz*, cette table occupe aussi un rôle central dans le modèle relationnel. Elle permet de faire le lien entre un quiz créé par un professeur et les réponses soumises à ce quiz par les étudiants. Elle possède donc une relation vers une table *Student* (encore une fois extérieure à l'application). De l'autre côté, cette table pointe vers *Quiz* et définit logiquement le quiz auquel l'étudiant a répondu. Un seul champ est présent : la date et l'heure de la soumission des réponses.

SqSubmit

Il s'agit simplement de la réponse apportée à une question simple. La table a donc un champ *text* qui contient la réponse soumise par l'élève. Elle possède aussi deux relations, une vers *SimpleQuestion* pour préciser la question auquel l'élève a répondu, et une autre vers *CompletedQuiz*. La réponse soumise par l'élève sera ensuite comparée à(aux) solution(s) enregistrées pour déterminer si les points sont attribués ou non.

QcmSubmitOne et QcmSubmitMulti

Ces deux tables sont très similaires. *QcmSubmitOne* contient une relation vers l'option sélectionnée par l'étudiant dans une question à choix multiples avec *multi_answers* valant `False`, tandis que *QcmSubmitMulti* peut contenir des relations vers plusieurs options, quand *multi_answers* vaut `True`. Il s'agit donc dans le premier cas d'une relation *many-to-one*, puique plusieurs lignes (réponses fournies par différents élèves) peuvent pointer vers la même option. Dans le deuxième cas, c'est une relation de type *many-to-many*, puisque plusieurs lignes peuvent pointer vers plusieurs options.

Dans Django, voici comment seront définies ces relations :

```
id_selected = models.ForeignKey(QcmChoice, null=True) #Relation many-to-one -> Une seule option chois
id_selected = models.ManyToManyField(QcmChoice, null=True) #Relation many-to-many -> Plusieurs optio
```

L'argument `null` vaut ici `True` car il se peut que l'étudiant ne coche aucun choix. Dans ce cas-là, il n'obtiendra dans tous les cas aucun point.